Escuela Politécnica Superior

Ingeniería Técnica en Informática de Gestión

Universidad Carlos III de Madrid

DISEÑO E IMPLEMENTACIÓN DE UN LABORATORIO DE OPENSTACK

Proyecto de Fin de Carrera

Autor del Proyecto: **Jesús Sánchez Manzanero**

Director del Proyecto **Jaime José García Reinoso**

Leganés, Septiembre 2015

AGRADECIMIENTOS

Me gustaría agradecer a todas las personas que han hecho que esto llegue a su fin después de muchos años de estudios, querría agradecerles en primer lugar a todos los profesores que he tenido a lo largo de los años, que han conseguido que yo llegue a este final, en especial a mi tutor Jaime Jose García Reinoso, porque nunca pensé que acabaría y el proyecto fin de carrera y gracias a él lo he conseguido.

También me gustaría darle las gracias a todos mis compañeros de estudio a lo largo de tantos años que me han ayudado siempre que han podido y que sin ellos no habría llegado, además me gustaría agradecerles también a todos mis compañeros de trabajo todo lo que he aprendido a lo largo de estos años y que me han hecho ser el profesional que soy ahora.

Y por último y más especial de todos a mi familia, en primer lugar a mis padres que han tenido la paciencia y el tesón para que yo termine mis estudios, y en especial a mi mujer Vanesa que ha hecho esfuerzos sobrehumanos cuidando a nuestros hijos para que yo tuviera tiempo para poder hacer este proyecto y sin ella y mis niños no hubiese tenido fuerzas para lograrlo.

INDICE

1º INTRODUCCIÓN.

Este proyecto está dirigido a la enseñanza de cloud computing. El "Cloud computing" es un nuevo modelo de prestación de servicios de negocio y tecnología, que permite incluso al usuario acceder a un catálogo de servicios estandarizados y responder con ellos a las necesidades de su negocio, de forma flexible y adaptativa, en caso de demandas no previsibles o de picos de trabajo, pagando únicamente por el consumo efectuado, o incluso gratuitamente en caso de proveedores que se financian mediante publicidad o de organizaciones sin ánimo de lucro.

El cambio que ofrece la computación desde la nube es que permite aumentar el número de servicios basados en la red. Esto genera beneficios tanto para los proveedores, que pueden ofrecer, de forma más rápida y eficiente, un mayor número de servicios, como para los usuarios que tienen la posibilidad de acceder a ellos, disfrutando de la 'transparencia' e inmediatez del sistema y de un modelo de pago por consumo. Así mismo, el consumidor ahorra los costes salariales o los costes en inversión económica (locales, material especializado, etc.).

1.1 MOTIVACIÓN.

La motivación que ha promovido este proyecto es la enseñanza a los alumnos de la universidad de un producto dirigido al cloud computing, en especial a la infraestructura tipo servicio (IaaS).

En nuestro caso elegimos openstack por un punto principal que lleva a la enseñanza, este es el de su licencia de software libre que nos permite poder utilizar todos sus módulos de forma gratuita y en el futuro nos podría servir para aportar nosotros nuevos conocimientos al producto, ya que este producto puede aceptar opiniones y contribuciones de cualquier usuario.

Al ser de software libre está orientado a adoptar estándares abiertos, lo que nos serviría para un futuro si se estandarizan las herramientas de cloud. Este proyecto está intentando esta estandarización y mientras tanto utiliza medios para poderse hablar con herramientas de distintas empresa como el cloud de Amazon.

Además que grandes empresas como Red hat, Dell, HP, IBM, Cisco, VMware y muchas más estén contribuyendo de forma gratuita con sus recursos al crecimiento de este software nos indica que tiene un gran futuro en el sector y que puede servir a los alumnos a encontrar futuros puestos de trabajos con este software, debido a su gran crecimiento y a su uso actual.

Por estas razones de estandarización, de software libre y de un futuro empresarial para los alumnos creemos que en un mundo donde el cloud computing está creciendo a marchas forzadas y es muy interesante que los nuevos ingenieros aprendan a utilizar una herramienta de tanto futuro y con un crecimiento tan grande como openstack.

1.2 OBJETIVOS.

Los objetivos principales de este proyecto es que los alumnos puedan poder instalar en cualquier infraestructura y con cualquier base una estructura de openstack para que pueda servir cloud público o privado a cualquier empresa o universidad.

La enseñanza de este laboratorio se dividirá en tres partes.

- Una primera que será la instalación de la base necesaria para nuestro proyecto, debido a que es de nivel formativo necesitaremos instalar el laboratorio con el software necesario para el funcionamiento de este y del producto openstack.
- Una segunda parte donde instalaremos y configuraremos los componentes básicos para el funcionamiento de openstack. En nuestro caso:
    - - Keystone.
    - - Glance
    - - Nova
    - - Neutron
    - - Cinder
    - - Horizon.
- Una tercera parte y final en la que generaremos un cloud privado básico donde se instalara un esquema tipo de empresa.

El objetivo principal es que los alumnos después de haber realizado todos los pasos necesarios para realizar estos 3 puntos sean capaces de poder instalar ellos solos un openstack en máquinas físicas para dar servicio de cloud.

2. ESTADO DEL ARTE.

Para poder definir este proyecto deberemos explicar los 3 puntos principales de este en primer caso explicaremos que es un Cloud y los tipos que hay, en segundo lugar explicaremos la herramienta que vamos a utilizar para el cloud computing (en nuestro caso openstack) y por último lugar y no menos importante explicaremos la virtualización, que es la base de openstack y del cloud computing.

2.1 CLOUD COMPUTING.

El cloud computing vamos a explicarlo en 4 puntos, primero definiéndolo, segundo diciendo sus características y por último vamos a definirlo por sus tipos de servicio y por los tipos de cloud existentes en la actualidad.

2.1.1 Definición.

El *cloud computing* es un modelo tecnológico que permite el acceso adaptado y bajo demanda en red a un conjunto compartido de recursos de computación configurables compartidos (por ejemplo: redes, servidores, equipos de almacenamiento, aplicaciones y servicios), que pueden ser rápidamente aprovisionados y liberados con un esfuerzo de gestión reducido o interacción mínima con el proveedor del servicio.

Otra definición complementaria es la aportada por el RAD Lab de la Universidad de Berkeley, desde donde se explica que el *cloud computing* se refiere tanto a las aplicaciones entregadas como servicio a través de Internet, como el hardware y el software de los centros de datos que proporcionan estos servicios. Los servicios anteriores han sido conocidos durante mucho tiempo como Software as a Service (SaaS), mientras que el hardware y software del centro de datos es a lo que se llama nube.

Según IBM: «El Cloud Computing es una categoría de soluciones de computación que permite a los usuarios acceder a recursos bajo demanda, según sus necesidades, ya sean físicos o virtuales, dedicados o compartidos y sin importar como se accede a ellos.»

Según el NIST: «es un modelo que proporciona de manera conveniente, acceso por demanda a un conjunto compartido y de recursos informáticos (redes, servidores, almacenamiento, aplicaciones, etc.) que pueden ser rápidamente dispuestos con un esfuerzo mínimo por parte del proveedor de estos recursos»

2.1.2 Características.

Las características del cloud computing son las siguientes:

- Servicio disponible de forma automática y a demanda: Un usuario puede comenzar a utilizar un recurso de cloud (almacenamiento, una instancia, etc.) sin ninguna intervención por parte de un operador de la empresa que presta el servicio.
- Acceso a través de la red: Los recursos están disponibles a través de la red (Internet u otro tipo de red pública o privada), mediante mecanismos estandarizados que permitan el uso de clientes diversos, desde teléfonos a grandes ordenadores.
- Los recursos se agrupan en pools en un modelo multi-tenancy. Los recursos son compartidos en general por múltiples clientes, que pueden disponer de ellos a demanda y a los que se les debe garantizar aislamiento y seguridad, a pesar de estar haciendo uso de recursos compartidos.
- Elasticidad: Es un concepto nuevo relacionado con el cloud y que lleva al extremo el concepto de escalabilidad, ya que los recursos del usuario del cloud pueden crecer y decrecer de forma rápida en función de sus necesidades.
- Pago por uso: La utilización real que hace el usuario de los recursos y no una tarificación por tramos es lo que define el pago de los servicios de cloud.

2.1.3 Modelos de servicios.

Los servicios que se ofrezcan a los usuarios que cumplan estos requisitos se pueden clasificar en función del tipo de servicio que ofrezcan en tres capas: SaaS, PaaS e IaaS.

- Software as a Service (SaaS): Aplicación completa ofrecida como servicio en la nube. Es el ejemplo más conocido y usado de cloud computing en el que cualquier usuario hace uso de un determinado software de una empresa a través de Internet, como por ejemplo los servicios de Google, microsoft Office 365 y un larguísimo etcétera.
- Platform as a Service (PaaS): Aplicación completa para el desarrollo y despliegue de software. Los desarrolladores de software pueden optar por utilizar una plataforma en la nube para sus desarrollos, facilitándoles mucho las tareas de pruebas y despliegue. Algunas de las opciones más conocidas de PaaS son Google App Engine, Windows Azure, Red Hat OpenShift o Heroku. Obviamente esta capa de cloud es sólo para desarrolladores o empresas que se dedican al desarrollo, pudiendo ser obviada por el resto de usuarios.
- Infrastructure as a Service (IaaS): Principalmente almacenamiento y capacidades de cómputo (máquinas virtuales) ofrecidos como servicio en la nube. Los administradores de sistemas de todo el mundo pueden hoy en día plantearse montar un servicio en una máquina física, una máquina virtual o una instancia en el cloud, ofreciendo esta última unas opciones muy interesantes sobre todo para despliegues de demanda variable. Los servicios de cloud IaaS más conocidos son los de Amazon Web Services, RackSpace Cloud, Joyent o Windows Azure.

2.1.4 Tipos de nubes.

Existen hoy en día 4 tipos de nubes en el mercado las cuales procedemos a explicar a continuación:

1.- Cloud Público (Externo)

Forma de implementación caracterizada por la oferta de servicios de computación virtualizados (bases de datos, sistemas operativos, plataformas de desarrollo, aplicaciones, etc.) por parte de los proveedores para múltiples clientes, accediendo éstos a dichos servicios a través de Internet o redes privadas virtuales (VPNs). Como características inherentes a esta forma de implementación podemos citar las que siguen:

- Reducido plazo de tiempo para la disponibilidad del servicio.

- No se requiere llevar a cabo inversión monetaria para su implementación.

- Permite la externalización a un proveedor de servicios *cloud* de todas las funciones básicas de la empresa.

- Posibilita el aprovechamiento de la infraestructura de los proveedores de servicios, permitiendo adicionalmente una alta escalabilidad y flexibilidad en la modificación del dimensionamiento del servicio.

- Favorece la utilización de conjuntos de software estándar.

- Lleva asociadas unas cuotas iniciales de pago más bajas que el resto de implementaciones. Adicionalmente los costes del *cloud* público son variables, cumpliendo el principio de pago por uso.

- La información corporativa se encuentra alojada en la nube pública junto a la del resto de clientes del proveedor, lo que implica, además de no poder tener localizada físicamente dicha información, imponer al proveedor una serie de requisitos de alta exigencia en temas de seguridad y protección de datos.

2.- Cloud Privado (Interno)

Forma de implementación caracterizada por el suministro por parte del proveedor, de entornos virtualizados que pueden ser implementados, usados y controlados por la misma empresa contratante del servicio. Esto indica no solo que la solución *cloud* puede ser administrada por la organización contratante, por el proveedor o por un tercer actor; sino que puede existir en las instalaciones propias del cliente o fuera de las mismas.

Como características propias de esta forma de implementación se enumeran las siguientes:

- Reducido plazo de tiempo para la puesta en servicio y una alta flexibilidad en la asignación de recursos.

- Al contrario que el *cloud público*, requiere de inversión económica para la implementación de la solución contratada.

- Lleva asociados sistemas y bases de datos locales.

- Ofrece la posibilidad de aprovechar el personal existente y las inversiones en sistemas de información realizadas con anterioridad.

- Implica más especificidad en la solución adquirida, ya que está diseñada para ajustarse a las necesidades propias de la empresa contratante.

- Permite disponer de un control total de la infraestructura, de los sistemas y de la información corporativa tratada por éstos.

- Facilita el control y la supervisión de los requisitos de seguridad y protección de la información almacenada.

3.- Cloud Híbrido

Forma de implementación cuya infraestructura *cloud* (en la nube) se caracteriza por aunar dos o más formas de *clouds* (privado, comunitario o público), los cuáles continúan siendo entidades únicas interconectadas mediante tecnología estandarizada o propietaria, tecnología que permite la portabilidad de datos y aplicaciones (ej. el rebalanceo de cargas entre nubes). Una entidad que emplee esta forma de implementación se podría beneficiar de las ventajas asociadas a cada tipo de *cloud*, disponiendo con ello de una serie de características adicionales, tal y como se muestra a continuación:

- Ofrece una mayor flexibilidad en la prestación de servicios de TI, al mismo tiempo que se mantiene un mayor control sobre los servicios de negocio y de datos.

- Con una solución de *cloud* híbrido, al igual que en los casos detallados anteriormente, se consigue una rápida puesta en servicio.

- Implica mayor complejidad en la integración de la solución *cloud*, como consecuencia de ser una solución que se compone de dos formas distintas de implementación de servicios en la nube.

- Permite integrar las mejores características de las dos formas de implementación *cloud*, en cuanto al control de los datos y a la gestión de las funciones básicas de la entidad.

- Posibilita la selección por parte del proveedor, de infraestructura escalable y flexible, permitiendo una alta agilidad en el redimensionamiento de la solución.

- Permite el control interno de los servicios *cloud* desde la propia entidad.

4.-Cloud de Comunidad

Se trata de *clouds* utilizados por distintas organizaciones cuyas funciones y servicios sean comunes, permitiendo con ello la colaboración entre grupos de interés.

Ejemplos de esta forma de implementación son los *clouds* de comunidades de servicios de salud (en inglés, *healthcare community cloud*) para facilitar el acceso aplicaciones e información crítica de carácter sanitario, y los *clouds* de comunidad gubernamentales (en inglés, *government community cloud*) para facilitar el acceso a recursos de interoperabilidad entre organismos públicos y Administraciones Públicas.

Al analizar un *cloud* de comunidad, se debe considerar que, en principio, sus fortalezas y debilidades se sitúan entre las del privado y las del público. En general, el conjunto de recursos disponibles con un *cloud* de comunidad es mayor que en el privado, con las ventajas evidentes que ello conlleva en términos de elasticidad. Sin embargo, la cantidad de recursos es menor que los existentes en una solución de *cloud* público, limitando la elasticidad respecto a dicho *cloud* público. Por otra parte, el número de usuarios de este tipo de nube es menor que los de la nube pública, lo que la dota de mayores prestaciones en cuestiones de seguridad y privacidad.

Figura 2. Modelo visual de NIST de la definición de trabajo de *Cloud Computing*

Fuente: CSA.

2.2 OPENSTACK.

2.2.1 Definición:

OpenStack es una colección de tecnologías Open Source que proporcionan un software para el despliegue escalable de un cloud computing. OpenStack proporciona Infraestructura como Servicio o IaaS (Infrastructure as a Service) y es un proyecto que se inició en el año 2010 por la empresa Rackspace Cloud y por la agencia espacial norteamericana, NASA. Actualmente más de 150 empresas se han unido al proyecto, entre las que se encuentran empresas tan importantes como AMD, Intel, Canonical, SUSE Linux, Red Hat, IBM, Dell, HP, Cisco, etc. OpenStack es software libre bajo los términos de la licencia Apache.

Actualmente OpenStack desarrolla dos proyectos relacionados: OpenStack Compute, que proporciona recursos computacionales a través de máquinas virtuales y gestión de la red, y OpenStack Object Storage, que proporciona un servicio de almacenamiento de objetos redundante y escalable. Muy relacionado con el proyecto "OpenStack Compute" tenemos otros proyectos complementarios como Keystone o Glance que describiremos en breve.

OpenStack puede ser utilizado por cualquiera organización que busque desplegar un cloud de gran escala tanto para uso privado como público. OpenStack es un proyecto interesante casi para cualquier tipo de organización: pequeñas y medianas empresas, administración, grandes corporaciones, proveedores de servicio, empresas de valor añadido, centros de cálculo y un largo etcétera

2.2.2 Componentes de OpenStack

Actualmente, hay siete componentes principales de OpenStack: Compute, Object Storage, Identity, Image Service, Dashboard, Network y Block Storage.

OpenStack Compute es el controlador de la estructura básica del Cloud. Es el encargado de iniciar las instancias (máquinas virtuales) de los usuarios y grupos. También es el servicio encargado de la gestión de la red virtual para cada instancia o para las múltiples instancias que formen parte de un proyecto (tenant).

OpenStack Object Storage es el servicio encargado del almacenamiento masivo de objetos a través de un sistema escalable, redundante y tolerante a fallos. Las posibles aplicaciones de Object Storage son numerosas, como por ejemplo: almacenamiento simple de ficheros, copias de seguridad, almacenamiento de streamings de audio/vídeo, almacenamiento secundario/terciario, desarrollo de nuevas aplicaciones con almacenamiento integrado, etc.

OpenStack Identity Service es un servicio usado para la autenticación entre el resto de componentes. Este servicio utiliza un sistema de autenticación basado en tokens y se incorporó en la versión 2012.1 de OpenStack.

OpenStack Image Service es un servicio para la búsqueda y recuperación de imágenes de máquinas virtuales. Este servicio puede almacenar las imágenes directamente o utilizar mecanismos más avanzados como: usar Object Storage como servicio de almacenamiento, usar Amazon's Simple Storage Solution (S3) directamente, o usar Object Storage como almacenamiento intermedio de S3.

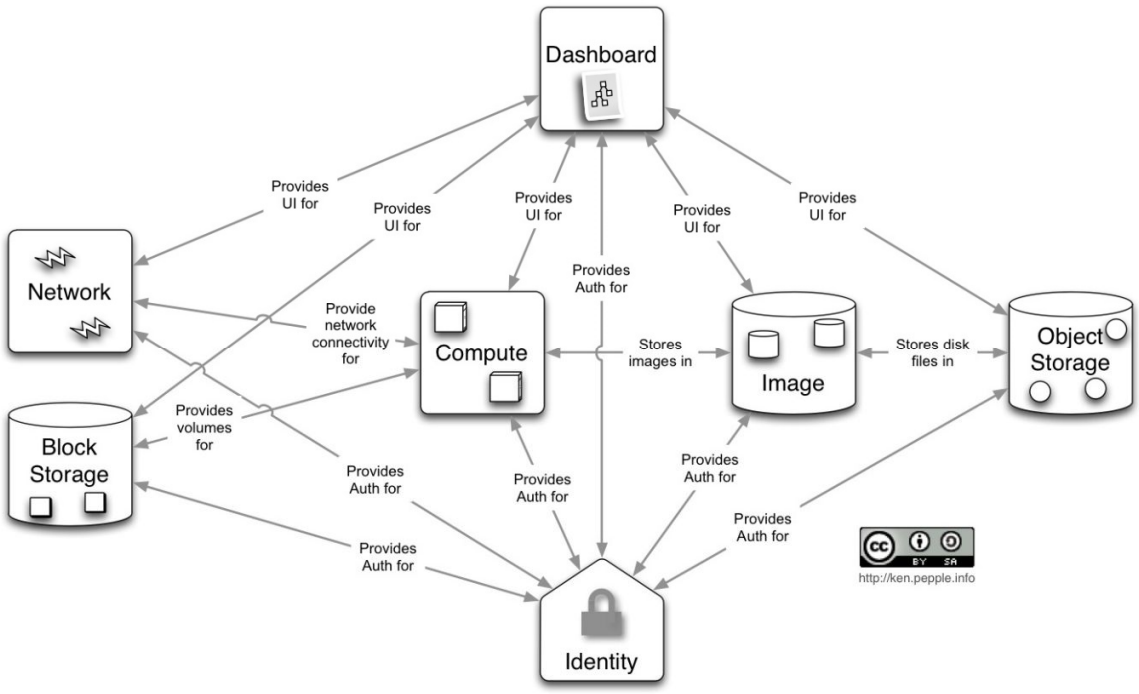
OpenStack Dashboard es un panel web para el manejo de instancias y volúmenes. Este servicio es realmente una aplicación web desarrollada en django que permite comunicarse con las diferentes APIs de OpenStack de una forma sencilla. OpenStack Dashboard es fundamental para usuarios noveles y en general para realizar acciones sencillas sobre las instancias

Openstack Network. El servicio Network, con nombre Neutron. El objetivo principal de Neutrón es proporcionar "conectividad de red como servicio" entre las interfaces de red gestionadas por otros servicios como Nova. Esto permitirá una gran flexibilidad a la hora de que los usuarios finales puedan crear sus propias redes e interconectar entre sí las instancias.

Openstack Block Storage. De forma complementaria al almacenamiento de objetos que realiza swift, este componente de nombre Cinder es el encargado del almacenamiento de bloques, que se utilizan en las instancias de OpenStack, es equivalente al servicio de pago Elastic Block Storage (EBS) de Amazon.

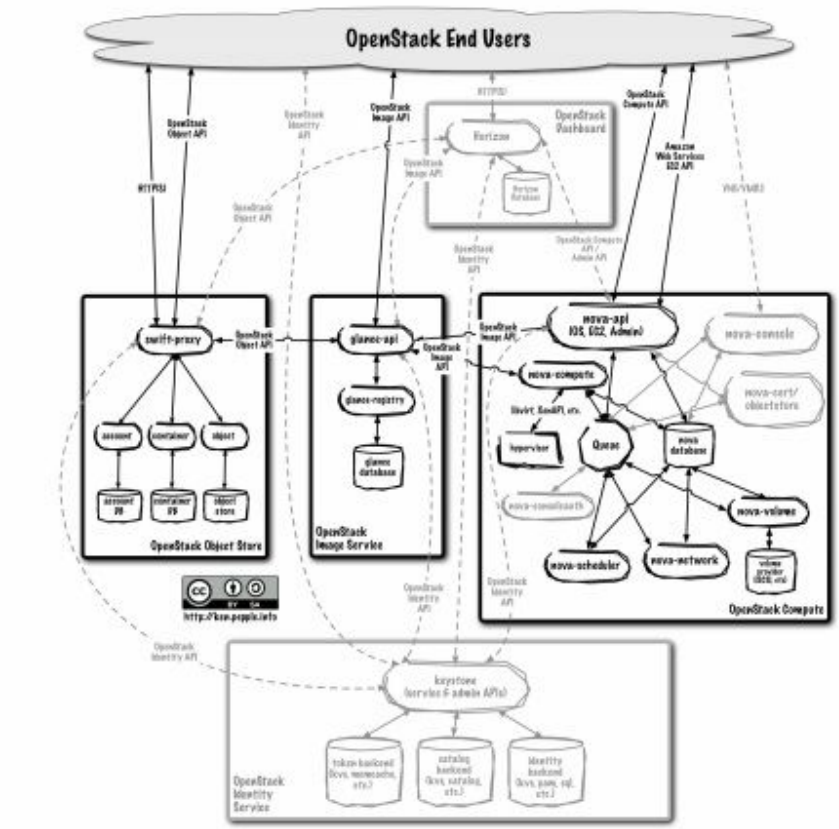
2.2.3 Arquitectura conceptual

Desde una perspectiva global, OpenStack está diseñado para "entregar un sistema operativo para el despliegue de clouds masivamente escalables". Para poder lograrlo, cada uno de los servicios que conforman OpenStack están diseñados para trabajar conjuntamente y poder proporcionar una Infraestructura como Servicio (IaaS, Infrastructure as a Service) completa. Esta integración se consigue a través de APIs (Application Programming Interfaces) que cada servicio ofrece, y que cada servicio puede consumir. Mientras que estas APIs permiten a cada uno de los servicios utilizar el resto, también permiten al desarrollador poder reemplazar cualquier servicio con otra implementación, siempre y cuando se respeten estas APIs. Dichas APIs también se encuentran disponibles para el usuario final del cloud.



2.2.4 Arquitectura lógica

Como nos podemos imaginar, la arquitectura real del cloud, su arquitectura lógica, es mucho más complicada que la mostrada anteriormente. Como cualquier arquitectura orientada a servicios, cualquier diagrama que intente ilustrar todas las posibles combinaciones de comunicación de servicios, enseguida se vuelve muy confuso. El siguiente diagrama trata de mostrar el escenario más común, mostrando arquitectura perfectamente integrada de un cloud basado en OpenStack.



Más adelante explicaremos más detenidamente los componentes que se van a instalar en éste proyecto y profundizaremos más en ellos.

2.3 VIRTUALIZACIÓN.

2.3.1 ¿Que es la virtualización?

Es la combinación de hardware y software que permite a un recurso físico funcionar como múltiples recursos lógicos,

En la virtualización existen dos tipos de máquinas:

- Anfitrión: sistema operativo que ejecuta la virtualización y controla el hardware físico. También llamados hipervisores.

- Huésped: Sistema operativo virtualizado, pueden existir de 1 a n anfitriones en el mismo host y el funcionamiento entre ellos no debe interferir, ni debe interferir con el anfitrión.

Los huéspedes también llamados Máquinas Virtuales permiten que distintos sistemas operativos, tareas y configuraciones convivan en la misma máquina y abstrae los recursos físicos de la anfitriona.

Tipos de hipervisores:

1. Nativo: El hipervisor es una capa entre el hardware y el sistema operativo.

2. Hosted: El hipervisor es una capa de software que corre sobre el sistema operativo anfitrión.

2.3.2 Tipos de virtualizadores

- Emulación

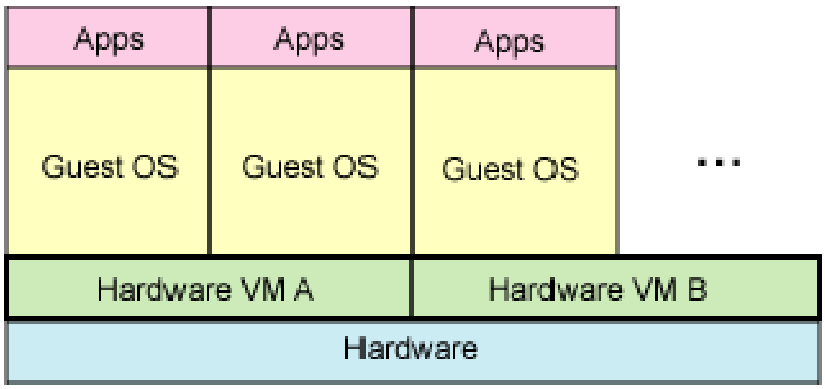
La emulación se basa en crear máquinas virtuales que emulan el hardware de una o varias plataformas hardware distintas. Este tipo de virtualización es la más costosa y la menos eficiente, ya que obliga a simular completamente el comportamiento de la plataforma hardware a emular e implica también que cada instrucción que se ejecute en estas plataformas sea traducida al hardware real.

Sin embargo la emulación tiene características interesantes, como poder ejecutar un sistema operativo diseñado para una plataforma concreta sobre otra plataforma, sin tener que modificarlo, o en el desarrollo de firmware para dispositivos hardware, donde se pueden comenzar estos desarrollos sin tener que esperar a tener disponible el hardware real.

Uno de los ejemplos más destacados de la actualidad es QEMU. QEMU, entre otras cosas, permite emular diferentes plataformas Hardware como x86, x86-64, PowerPC, SPARC o MIPS. Así pues, podríamos tener dentro de un servidor linux varios equipos x86 o PowerPC, corriendo diferentes versiones de Linux.

Ejemplos:

- **Bochs**: http://bochs.sourceforge.net/
- **MAME**: http://mamedev.org/
- **QEMU**: http://bellard.org/qemu/



- Virtualización completa

Con este término se denominan aquellas soluciones que permiten ejecutar sistemas operativos huésped (Guest), sin tener que modificarlos, sobre un sistema anfitrión (Host), utilizando en medio un Hypervisor o Virtual Machine Monitor que permite compartir el hardware real. Esta capa intermedia es la encargada de monitorizar los sistemas huésped con el fin de capturar determinadas instrucciones protegidas de acceso al hardware, que no pueden realizar de forma nativa al no tener acceso directo a él.

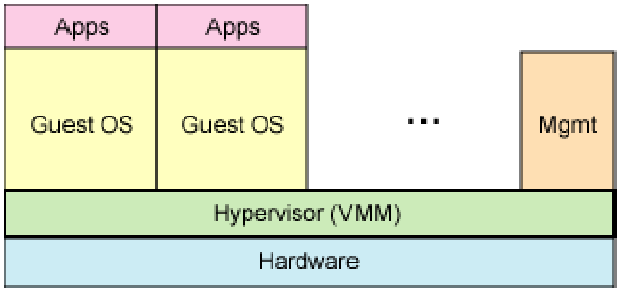
Su principal ventaja es que los sistemas operativos pueden ejecutarse sin ninguna modificación sobre la plataforma, aunque como inconveniente frente a la emulación, el sistema operativo debe estar soportado en la arquitectura virtualizada.

En lo que respecta al rendimiento, éste es significativamente mayor que en la emulación, pero menor que en una plataforma nativa, debido a la monitorización y la mediación del hypervisor. Sin embargo, recientes incorporaciones técnicas en las plataformas x86 hechas por Intel y AMD, como son Intel VT y AMD-V, han permitido que soluciones basadas en la virtualización completa se acerquen prácticamente al rendimiento nativo.

Hay que tener en cuenta también que la virtualización completa no se refiere a todo el conjunto de hardware disponible en un equipo, sino a sus componentes principales, básicamente el procesador y memoria. De esta forma, otros periféricos como tarjetas gráficas, de red o de sonido, no se virtualizan. Las máquinas huésped no disponen de los mismos dispositivos que el anfitrión, sino de otros virtuales genéricos. Por ejemplo, si se dispone de una tarjeta nVidia GeForce en el anfitrión, los equipos huésped no verán esta tarjeta sino una genérica Cirrus.

Ejemplos:

- **VirtualBox**: http://www.virtualbox.org/
- **VMWare**: http://www.vmware.com/
- **KVM**: http://kvm.qumranet.com/



- Paravirtualización.

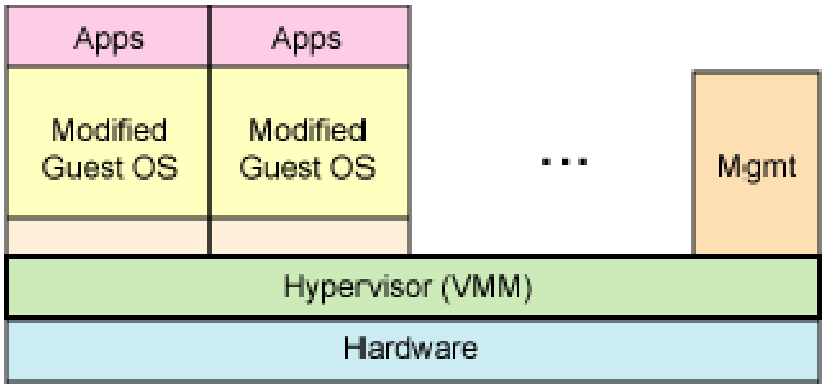
La paravirtualización surgió como una forma de mejorar la eficiencia de las máquinas virtuales y acercarlo al rendimiento nativo. Para ello se basa en que los sistemas virtualizados (huésped) deben estar basados en sistemas operativos especialmente modificados para ejecutarse sobre un Hypervisor. De esta forma no es necesario que éste monitorice todas las instrucciones, sino que los sistemas operativos huésped y anfitrión colaboran en la tarea.

Uno de los componentes más destacados de esta familia es XEN, el cual fue mi principal candidato durante bastante tiempo. Permite paravirtualización utilizando sistemas operativos modificados, y virtualización completa sobre procesadores con tecnología Intel-VT o AMD-V. Para la gestión de las máquinas virtuales existen aplicaciones propietarias e incluso alguna open-source como ConVirt, que permite gestionar también desde un único sitio las máquinas virtuales de diferentes servidores, realizar tareas sobre ellas, o modificar sus configuraciones.

Cabría destacar otro tipo de productos para virtualizar que son aplicaciones de escritorio para virtualizar (OS level Virtualization), como Vmware-player y VirtualBox. Estos tipos de virtualización están muy bien para virtualizar Sistemas Operativos en nuestro escritorio y hacer pruebas puntuales pero no para estar siempre ejecutándose. Ya que son programas que requieren tener las X (Desktop), comparten los recursos del Host y no están indicados para ser ejecutados con un servicio o demonio del sistema en un servidor y su rendimiento es menor.

Ejemplos:

- **User-mode Linux**: http://user-mode-linux.sourceforge.net/
- **Xen**: http://www.xen.org/



- Virtualización a nivel de sistema operativo.

La virtualización se hace instanciando la imagen del sistema operativo sin hipervisor. El sistema operativo esta modificado para permitir múltiples procesos en diferentes espacios de usuario aislados unos de otros, cada uno con su configuración de red Ej: LXC, zonas (opensolaris)

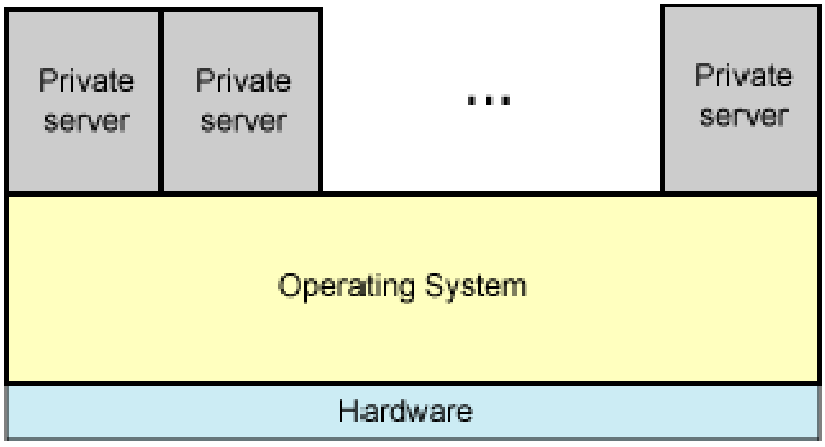
Una arquitectura alternativa para la máquina virtual es instalar una capa de virtualización encima del sistema operativo host. Este sistema operativo host es responsable de administrar el hardware. Los sistemas operativos invitados se instalan encima del nivel de virtualización. Es posible ejecutar aplicaciones especializadas en las máquinas virtuales. También existen otras aplicaciones que se pueden ejecutar directamente con el sistema operativo host.

Esta arquitectura con un host tiene algunas ventajas marcadas. Primero, el usuario puede instalar esta arquitectura de máquina virtual sin modificar el sistema operativo host. El software de virtualización puede descansar en el sistema operativo host para entregar los controladores de dispositivos y otros servicios de bajo nivel. Esto simplifica el diseño de la máquina virtual y facilita la implementación.

Segundo, el método con un host es atractivo para muchas configuraciones de máquinas host. Comparado con la arquitectura de hypervisor o VMM, el rendimiento de la arquitectura con un host también puede ser bajo. Cuando una aplicación solicita acceso al hardware involucra cuatro niveles de asignación, lo que reduce el rendimiento significativamente. Cuando el Internet Security and Acceleration (ISA) de un sistema operativo invitado es diferente del ISA del hardware subyacente hay que realizar una traducción binaria. Aunque la arquitectura con un host es flexible, el rendimiento es demasiado lento para poder usarlo en la práctica.

Ejemplos:

- **Linux-VServer**: http://www.linux-vserver.org/
- **LXC**: http://lxc.sourceforge.net/
- **OpenVZ**: http://www.openvz.org/



3 DISEÑO E IMPLEMENTACIÓN

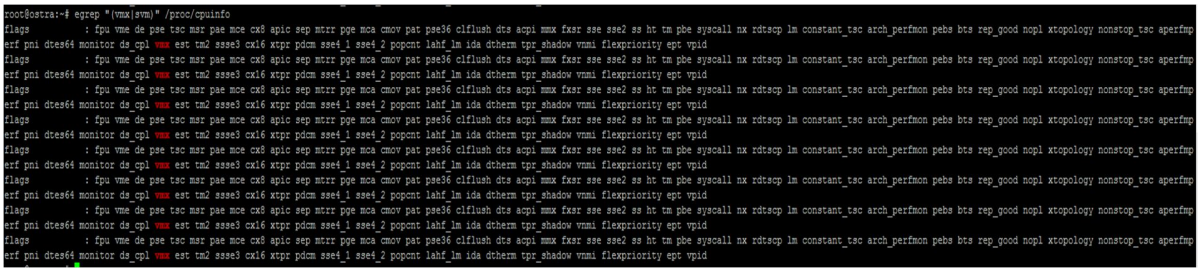
Debido a su carácter didáctico vamos a realizar una implementación en un solo ordenador, como podemos realizar esto, pues de una forma sencilla que nos va a poder realizar una estructura de empresa en una sola máquina a través de la virtualización.

Para esta virtualización tenemos que instalar sobre esta máquina física un sistema operativo. ¿Qué sistema operativo instalaremos?, pues como hemos dicho por el carácter didáctico utilizaremos un linux, ¿y qué linux instalaremos?, en nuestro caso hemos elegido ubuntu 14.04. Hemos elegido este sistema operativo porque contiene los paquetes necesarios para openstack, en otros sistemas tendremos que buscar repositorios externos o compilar determinados paquetes, lo que nos retrasaría y eliminaría el carácter didáctico de openstack de la práctica y ya pasaría a un carácter didáctico de linux que no es lo que buscamos.

En primer lugar para poder realizar el laboratorio tenemos que comprobar que la máquina donde se va a instalar es compatible con virtualización:

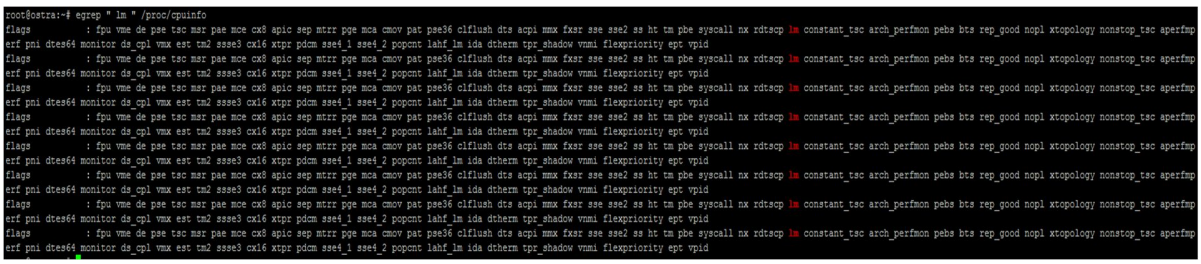
Comprobación de flags (virtualización)

egrep "(vmx|svm)" /proc/cpuinfo



Comprobación de flags (64 bits)

egrep " lm " /proc/cpuinfo



Comprobación del kernel (64 bits)

uname –m

```
root@ostra:~# uname -m
x86_64
```

Una vez comprobado que nuestra máquina es compatible con la virtualización tendremos que decidir sobre que virtualizador vamos a trabajar. El primer punto es que tipo de virtualización es más completa y más utilizada, viendo los tipos explicados anteriormente, la virtualización completa es la que más nos interesa debido a su mayor rendimiento y su mayor uso en el mercado. Entre los virtualizadores que existen en este tipo nos vamos a decantar por el kvm, ya que este es gratuito y de un gran uso en el mercado, lo que nos hace que sea el más interesante para este proyecto. Ahora daré una pequeña definición de kvm:

KVM es una solución de virtualización para Linux en hardware x86 (incluyendo hardware de 64-bits) que contienen las extensiones de virtualización Intel VT o AMD-V. Se compone de un módulo del kernel que puede ser cargado, kvm.ko, para proveer toda la infraestructura de virtualización base y un módulo específico del procesador, kvm-intel.ko o kvm-amd.ko.

Ya definido el virtualizador a utilizar debemos definir las máquinas y redes que vamos a utilizar en el laboratorio para montar openstack.

- Máquinas.

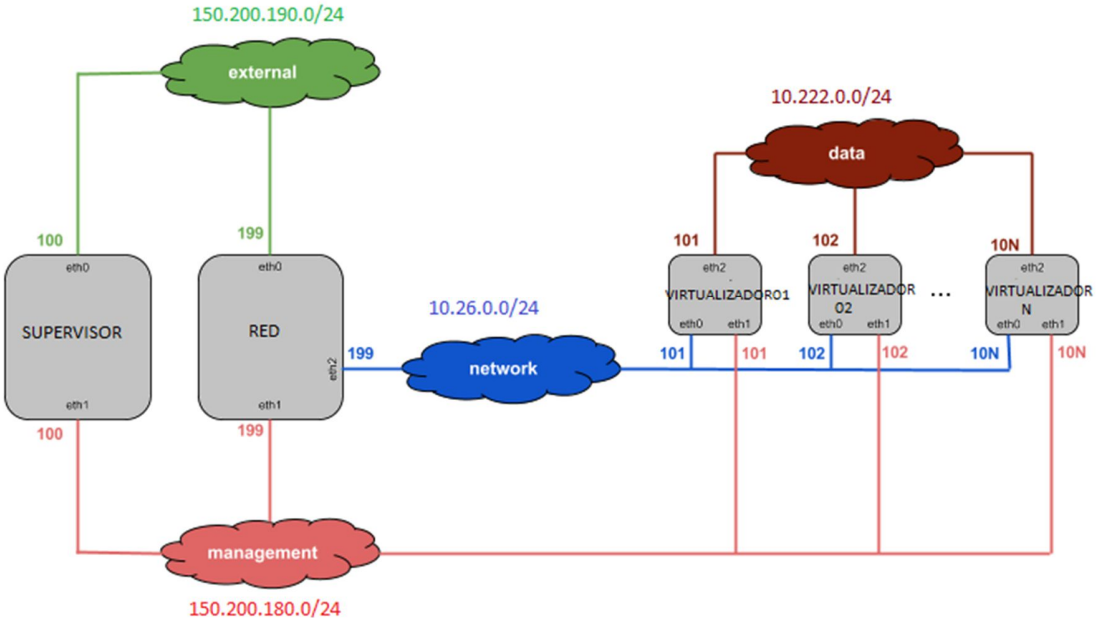
    1. Máquina donde residirá el conjunto principal de servicios de openstack, esta máquina es el corazón del sistema y donde irán la gran mayoría de los servicios de openstack (Supervisor).
    2. Máquina de red. Aunque podría residir en la máquina anterior al ser un servicio principal del proyecto hemos decidió sacarla para hacerla independiente a fallos de la máquina principal y así poder separar pesos (Red).
    3. Máquinas donde se crearán esas máquinas del cloud. Es decir, las máquinas virtualizadoras, allí correrán nuestras máquinas del cloud. (Virtualizador01 y Virtualizador02).

- Redes.

    En este punto podríamos haber decidido mil combinaciones, nosotros hemos decidido la siguiente:

    1. Red externa para poder tener conectividad con el exterior (150.200.190.0/24)
    2. Red de comunicación para hablar con neutrón. (10.26.0.0/24)
    3. Red de gestión, por aquí si hablarán los servicios. (150.200.180.0/24)
    4. Red de data, está la hemos generado por si los virtualizadores necesitaran almacenamiento por ISCSI y para uso del cloud (10.222.0.0/24)

A continuación pondremos un gráfico con la estructura del laboratorio.



3.1 CREACIÓN DEL LABORATORIO.

Una vez comprobado esto procedemos a instalar los paquetes necesarios para virtualizar con kvm:

apt-get install qemu-kvm libvirt-bin bridge-utils virt-viewer

Cuando esté instalado añadimos un usuario al grupo libvirtd:

adduser <usuario> libvirtd

Comprobaciones de que está instalado

virsh list o kvm-ok (este último solo en ubuntu)

Para poder gestionar el kvm deberemos instalar el Virtual Machine Manager, que es el gestor gráfico de gestión de kvm.

apt-get install virt-manager

Vamos a crear el pool donde residirán las máquinas de este laboratorio para esto tendremos que crearle un directorio :

mkdir /repositorio

Y cuando este creado a través de sistema operativo habrá que crearlo de nuevo pero esta vez en el kvm:

virsh pool-define-as --name vms --type dir --target /repositorio

Para comprobar los pools definidos:

virsh pool-list --all

Para arrancar el pool ejecutaremos:

virsh pool-start vms

Y para dejarlo con auto arranque:

virsh pool-autostart vms

El siguiente paso es crear un volumen dentro del pool creado, esto se realiza en el kvm, en nuestro caso crearemos uno básico de 5Gb, para realizar en él nuestra máquina base::

virsh vol-create-as --pool vms --name ubuntu1404.img --capacity 5G --format qcow2

Para listar los volúmenes creados ejecutaremos:

virsh vol-list --pool vms

3.1.1 Creación de Redes virtuales.

Para ello primero definiremos las redes que necesitamos, en nuestro caso las cuatro redes definidas en el esquema anterior. Todas ellas las definiremos a través de ficheros xml, principalmente lo que definiremos en estas redes son sus rangos, les daremos un dhcp y definiremos las ips de las máquinas como en el esquema anteriormente expuesto:

- *Red extenal:*

FICHERO NET-EXTERNAL.XML:

```
<network>
  <name>external</name>
  <bridge name='virbr10' />
  <forward mode='nat'/>
  <mac address='00:00:AC:10:00:FE'/>
  <ip address='150.200.190.254' netmask='255.255.255.0'>
    <dhcp>
      <range start='150.200.190.250' end='150.200.190.253' />
      <host mac='00:00:AC:10:00:64' name='supervisor.external' ip='150.200.190.100' />
      <host mac='00:00:AC:10:00:C7' name='red.external' ip='150.200.190.199' />
    </dhcp>
  </ip>
</network>
```

Para definir la red a través de kvm:

virsh net-define net-external.xml

Comprobaremos las redes definidas con:

virsh net-list --all

Ahora procederemos a arrancar la red:

virsh net-start external

Y después de arrancarla procederemos a dejarla en auto start:

virsh net-autostart external

- *Red Management:*

FICHERO NET-MANAGEMENT.XML

```
<network>
  <name>management</name>
  <bridge name="virbr11" />
  <mac address="00:00:0A:10:00:FE"/>
  <ip address="150.200.180.254" netmask="255.255.255.0">
    <dhcp>
      <range start="150.200.180.250" end="150.200.180.253" />
      <host mac="00:00:0A:10:00:64" name="supervisor.management"
ip="150.200.180.100" />
      <host mac="00:00:0A:10:00:C7" name="red.management" ip="150.200.180.199" />
      <host mac="00:00:0A:10:00:65" name="virtualizador01.management"
ip="150.200.180.101" />
      <host mac="00:00:0A:10:00:66" name="virtualizador02.management"
ip="150.200.180.102" />
    </dhcp>
  </ip>
</network>
```

Procederemos a definirla, arrancarla y configurar su autoarranque con los siguientes 3 comandos.

```
virsh net-define net-management.xml
virsh net-start management
virsh net-autostart management
```

- *Red Network:*

FICHERO NET-NETWORK.XML

```
<network>
  <name>network</name>
  <bridge name="virbr12" />
  <mac address="10:00:C0:A8:00:FE"/>
  <ip address="10.26.0.254" netmask="255.255.255.0">
    <dhcp>
      <range start="10.26.0.250" end="10.26.0.253" />
      <host mac="10:00:C0:A8:00:65" name="virtualizador01.network" ip="10.26.0.101" />
      <host mac="10:00:C0:A8:00:66" name="virtualizador02.network" ip="10.26.0.102" />
      <host mac="10:00:C0:A8:00:C7" name="red.network" ip="10.26.0.199" />
    </dhcp>
  </ip>
</network>
```

Procederemos a definirla, arrancarla y configurar su autoarranque con los siguientes 3 comandos.

```
virsh net-define net-network.xml
virsh net-start network
virsh net-autostart network
```

- *Red Data:*

FICHERO NET-DATA.XML

```
<network>
  <name>data</name>
  <bridge name="virbr13" />
  <mac address="00:10:AC:10:64:FE" />
  <ip address="10.222.0.254" netmask="255.255.255.0">
    <dhcp>
      <range start="10.222.0.250" end="10.222.0.253" />
      <host mac="00:10:AC:10:64:65" name="virtualizador01.data" ip="10.222.0.101" />
      <host mac="00:10:AC:10:64:66" name="virtualizador02.data" ip="10.222.0.102" />
    </dhcp>
  </ip>
</network>
```

Procederemos a definirla, arrancarla y configurar su autoarranque con los siguientes 3 comandos.

```
virsh net-define net-data.xml
virsh net-start data
virsh net-autostart data
```

3.1.2. Creación Máquina Base.

El siguiente paso es la creación de una máquina virtual original la cual clonaremos para realizar las 4 máquinas virtuales que necesitamos, para eso bajaremos una iso del sistema operativo que necesitamos ubuntu-14.04.

```
wget http://ftp.udc.es/ubuntu-releases/14.04/ubuntu-14.04-server-amd64.iso
```

Y lo moveremos a nuestro repositorio.

```
mv ubuntu-14.04-server-amd64.iso /repositorio/
```

Hemos decidido esta imagen y este sistema operativo debido a que este sistema operativo tiene en su repositorio, todos los componentes de openstack y en nuestro caso de la versión Juno que es la que vamos a instalar.

Ahora procederemos a crear la máquina virtual a partir del siguiente fichero xml, en este xml le indicaremos las características de la máquina, memorias (2GB), cpus (2), el disco (que hemos definido antes) y lo principal le montaremos la imagen que hemos bajado antes para poder instalarla:

FICHERO VM-UBUNTU1404.XML

```
<domain type="kvm">
  <name>ubuntu1404</name>
  <memory unit="M">2048</memory>
  <vcpu>2</vcpu>
  <os>
    <type arch="x86_64">hvm</type>
    <boot dev="cdrom"/>
    <boot dev="hd"/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type="file" device="disk">
      <driver name="qemu" type="qcow2"/>
      <source file="/repositorio/ubuntu1404.img" />
      <target dev="vda" bus="virtio" />
    </disk>
    <disk type="file" device="cdrom">
      <driver name="qemu" type="raw"/>
      <source file="/repositorio/ubuntu-14.04-server-amd64.iso"/>
      <target dev="hdc" bus="ide"/>
    </disk>
    <interface type="network">
      <source network="external"/>
      <model type="virtio"/>
    </interface>
    <graphics type="vnc" port="-1" />
  </devices>
</domain>
```

La máquina virtual la crearemos con el siguiente comando:

```
virsh define vm-ubuntu1404.xml
```

Para ver todas las máquinas virtuales definidas ejecutaremos.

```
virsh list --all
```

Procederemos a arrancarla con el comando:

virsh start ubuntu1404

Una vez arrancada la definiremos con autoarranque.

virsh autostart ubuntu1404

Para poder instalar el sistema operativo por defecto de ubuntu nos conectaremos a la máquina virtual a través del siguiente comando:

virt-viewer ubuntu1404

Una vez instalada el sistema operativo ubuntu procederemos a expulsar el cd de la máquina virtual:

virsh change-media ubuntu1404 hdc --eject

Y una vez expulsado procederemos a borrar el dispositivo:

virsh detach-disk ubuntu1404 hdc --config

3.1.3 Creación máquinas laboratorio.

Ya instalada la máquina básica procederemos a instalar las 4 máquinas que necesitamos para el laboratorio.

Una vez definidas las redes necesarias para el laboratorio procederemos a clonar la máquina que hemos instalado con un sistema operativo ubuntu a las 4 máquinas necesarias para el laboratorio.

```
virsh vol-clone --pool vms ubuntu1404.img supervisor.img
virsh vol-clone --pool vms ubuntu1404.img red.img
virsh vol-clone --pool vms ubuntu1404.img virtualizador01.img
virsh vol-clone --pool vms ubuntu1404.img virtualizador02.img
```

A partir del clonado de los discos configuraremos estas 4 máquinas con las cosas necesarias para cada máquina.

- *Máquina SUPERVISOR:*

Crearemos la máquina virtual a través de un xml, en este xml le indicaremos las características de la máquina, memorias (3GB, le ponemos bastante ya que esta máquina es el corazón del proyecto y necesita más potencia), cpus (2), el disco (que hemos clonado en el punto anterior) y le definiremos las redes que necesarias para esta máquina (external y management):

FICHERO VM-SUPERVISOR.XML

```
<domain type="kvm">
  <name>supervisor</name>
  <memory unit="M">3072</memory>
  <vcpu>2</vcpu>
  <os>
    <type arch="x86_64">hvm</type>
    <boot dev="hd" />
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type="file" device="disk">
      <driver name="qemu" type="qcow2" />
      <source file="/repositorio/supervisor.img" />
      <target dev="vda" bus="virtio" />
    </disk>
    <interface type="network">
      <source network="external" />
      <model type="virtio" />
      <mac address="00:00:AC:10:00:64" />
    </interface>
    <interface type="network">
      <source network="management" />
      <model type="virtio" />
      <mac address="00:00:0A:00:00:64" />
    </interface>
    <graphics type="vnc" port="-1" />
  </devices>
</domain>
```

Procedemos a crear la máquina virtual y a arrancarla.

```
virsh define vm-supervisor.xml
virsh start supervisor
```

Una vez arrancada procedemos a configurarla desde el interior.
Procederemos a regenerar las claves ssh de la máquina.

```
rm /etc/ssh/*host*key*
dpkg-reconfigure openssh-server
```

Después de esto configuraremos sus redes en el fichero /etc/network/interfaces

```
auto eth0
iface eth0 inet static
  address 150.200.190.100
  netmask 255.255.255.0
  gateway 150.200.190.254
  dns-nameservers 150.200.190.254

auto eth1
iface eth1 inet static
  address 150.200.180.100
  netmask 255.255.255.0
```

Reiniciaremos la máquina virtual para que coja los cambios.

- *Máquina RED:*

Crearemos la máquina virtual a través de un xml, en este xml le indicaremos las características de la máquina, memorias (512MB, esta máquina no necesita tanto y por eso le ponemos un valor bajo para aligerar la carga del anfitrión), cpus (1), el disco (que hemos clonado en el punto anterior) y le definiremos las redes que necesarias para esta máquina (extenal, network y management):

FICHERO VM-RED.XML

```
<domain type="kvm">
  <name>red</name>
  <memory unit="M">512</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch="x86_64">hvm</type>
    <boot dev="hd"/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type="file" device="disk">
      <driver name="qemu" type="qcow2" />
      <source file="/repositorio/red.img" />
      <target dev="vda" bus="virtio" />
    </disk>
    <interface type="network">
      <source network="external" />
      <model type="virtio" />
      <mac address="00:00:AC:10:00:C7" />
    </interface>
    <interface type="network">
      <source network="management" />
      <model type="virtio" />
      <mac address="00:00:0A:00:00:C7" />
    </interface>
    <interface type="network">
      <source network="network" />
      <model type="virtio" />
      <mac address="00:00:C0:A8:00:C7" />
    </interface>
    <graphics type="vnc" port="-1" />
  </devices>
</domain>
```

Procedemos a crear la máquina virtual y a arrancarla.

```
virsh define vm-red.xml
virsh start red
```

Una vez arrancada procedemos a configurarla desde el interior.
Procederemos a regenerar las claves ssh de la máquina.

```
rm /etc/ssh/*host*key*
dpkg-reconfigure openssh-server
```

Después de esto configuraremos sus redes en el fichero /etc/network/interfaces

```
auto eth0
iface eth0 inet static
  address 150.200.190.199
  netmask 255.255.255.0
  gateway 150.200.190.254
  dns-nameservers 150.200.190.254

auto eth1
iface eth1 inet static
  address 150.200.180.199
  netmask 255.255.255.0

auto eth2
iface eth2 inet static
  address 10.26.0.199
  netmask 255.255.255.0
```

Reiniciaremos la máquina virtual para que coja los cambios.

- *Máquina VIRTUALIZADOR01:*

Crearemos la máquina virtual a través de un xml, en este xml le indicaremos las características de la máquina, memorias (2GB, le ponemos bastante porque estas máquinas la necesitaran para virtualizar), cpus (2), el disco (que hemos clonado en el punto anterior) y le definiremos las redes que necesarias para esta máquina (network, management y data):

FICHERO VM-VIRTUALIZADOR01.XML

```
<domain type="kvm">
  <name>virtualizador01</name>
  <memory unit="M">2048</memory>
  <vcpu>2</vcpu>
  <os>
    <type arch="x86_64">hvm</type>
    <boot dev="hd"/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type="file" device="disk">
      <driver name="qemu" type="qcow2" />
      <source file="/repositorio/virtualizador01.img" />
      <target dev="vda" bus="virtio" />
    </disk>
    <interface type="network">
      <source network="network" />
      <model type="virtio" />
      <mac address="00:00:C0:A8:00:65" />
    </interface>
    <interface type="network">
      <source network="management" />
      <model type="virtio" />
      <mac address="00:00:0A:00:00:65" />
    </interface>
    <interface type="network">
      <source network="data" />
      <model type="virtio" />
      <mac address="00:00:AC:10:64:65" />
    </interface>
    <graphics type="vnc" port="-1" />
  </devices>
</domain>
```

Procedemos a crear la máquina virtual y a arrancarla.

```
virsh define vm-virtualizador01.xml
virsh start virtualizador01
```

Una vez arrancada procedemos a configurarla desde el interior.
Procederemos a regenerar las claves ssh de la máquina.

```
rm /etc/ssh/*host*key*
dpkg-reconfigure openssh-server
```

Después de esto configuraremos sus redes en el fichero /etc/network/interfaces

```
auto eth0
iface eth0 inet static
  address 10.26.0.101
  netmask 255.255.255.0

auto eth1
iface eth1 inet static
  address 150.200.180.101
  netmask 255.255.255.0

auto eth2
iface eth2 inet static
  address 10.222.0.101
  netmask 255.255.255.0
```

Reiniciaremos la máquina virtual para que coja los cambios.

- *Máquina VIRTUALIZADOR02:*

Crearemos la máquina virtual a través de un xml, en este xml le indicaremos las características de la máquina, memorias (2GB, le ponemos bastante porque estas máquinas la necesitaran para virtualizar), cpus (2), el disco (que hemos clonado en el punto anterior) y le definiremos las redes que necesarias para esta máquina (network, management y data):

FICHERO VM-VIRTUALIZADOR02.XML

```
<domain type=”kvm”>
  <name>virtualizador02</name>
  <memory unit=”M”>2048</memory>
  <vcpu>2</vcpu>
  <os>
    <type arch=”x86_64”>hvm</type>
    <boot dev=”hd”/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type=”file” device=”disk”>
      <driver name=”qemu” type=”qcow2” />
      <source file=”/repositorio/virtualizador02.img” />
      <target dev=”vda” bus=”virtio” />
    </disk>
    <interface type=”network”>
      <source network=”network” />
      <model type=”virtio” />
      <mac address=”00:00:C0:A8:00:66” />
    </interface>
    <interface type=”network”>
      <source network=”management” />
      <model type=”virtio” />
      <mac address=”00:00:0A:00:00:66” />
    </interface>
    <interface type=”network”>
      <source network=”data” />
      <model type=”virtio” />
      <mac address=”00:00:AC:10:64:66” />
    </interface>
    <graphics type=”vnc” port=”-1” />
  </devices>
</domain>
```

Procedemos a crear la máquina virtual y a arrancarla.

```
Virsh define vm-virtualizador02.xml
virsh start virtualizador02
```

Una vez arrancada procedemos a configurarla desde el interior.
Procederemos a regenerar las claves ssh de la máquina.

```
Rm /etc/ssh/*host*key*
dpkg-reconfigure openssh-server
```

Después de esto configuraremos sus redes en el fichero /etc/network/interfaces

```
auto eth0
iface eth0 inet static
  address 10.26.0.102
  netmask 255.255.255.0

auto eth1
iface eth1 inet static
  address 150.200.180.102
  netmask 255.255.255.0

auto eth2
iface eth2 inet static
  address 10.222.0.102
  netmask 255.255.255.0
```

Reiniciaremos la máquina virtual para que coja los cambios.

Aunque según el dibujo las máquinas virtualizador01 y virtualizador02 no tienen conexión externa para la configuración del resto de la práctica necesitaremos que la tengan para la instalación y actualización de paquetes. Por esta razón deberemos hacer lo siguiente para darles esta red durante la instalación, red que deberá quitarse cuando acabe esta. Para ello deberemos apagar las máquinas.

```
Virsh shutdown virtualizador01
virsh shutdown virtualizador02
```

Añadirles unos interfaces de red a través de 2 ficheros xml:

FICHERO VIRTUALIZADOR01-EXT.XML

```
<interface type='network'>
  <source network='external'/>
  <mac address="00:00:AC:10:00:65" />
  <model type='virtio'/>
</interface>
```

Procederemos a darle la tarjeta de red correspondiente:

```
virsh attach-device virtualizador01 virtualizador01-ext.xml –config
```

FICHERO VIRTUALIZADOR02-EXT.XML

```
<interface type='network'>
  <source network='external'/>
  <mac address="00:00:AC:10:00:65" />
  <model type='virtio'/>
</interface>
```

Procederemos a darle la tarjeta de red correspondiente:

```
virsh attach-device virtualizador02 virtualizador02-ext.xml –config
```

Después de darle las tarjetas procederemos a encender las máquinas:

```
virsh start virtualizador01

virsh start virtualizador02
```

Una vez levantadas procederemos a solicitar nuestra IP de dhcp dentro de ambas máquinas:

```
dhclient eth3
```

Por último paso y solo por comodidad del trabajo definiremos en nuestro host las 4 máquinas virtuales por su red de management.

```
Echo –e "150.200.180.100\tsupervisor" >> /etc/hosts
echo –e "150.200.180.199\tred" >> /etc/hosts
echo –e "150.200.180.101\tvirtualizador01" >> /etc/hosts
echo –e "10.0.0.102\tvirtualizador02" >> /etc/hosts
```

Para poder utilizar cinder en el futuro primero tendremos que darle almacenamiento a los dos host virtualizador01 y virtualizador02 a través de kvm.

- Creamos el espacio que utilizaremos para meter las futuras máquinas virtuales creadas por openstack:

virsh vol-create-as --pool vms --name virtualizador01-datos01.img --capacity 25G --format qcow2
virsh vol-create-as --pool vms --name virtualizador02-datos01.img --capacity 25G --format qcow2

- Damos el espacio dado a las máquinas:

virsh attach-disk virtualizador01 /repositorio/virtualizador01-datos01.img --target vdb --driver qemu --subdriver qcow2 --config --live
virsh attach-disk virtualizador02 /repositorio/virtualizador02-datos01.img --target vdb --driver qemu --subdriver qcow2 --config --live

Una vez dado el espacio procederemos a instalar el lvm, para poder gestionar este espacio en la virtualización

apt-get install lvm2

Y crearemos un volumen en cada máquina para que tengan espacio de almacenamiento para las futuras máquinas virtuales:

pvcreate /dev/vdb
vgcreate cinder-volumes /dev/vdb

3.2 INSTALACIÓN PREVIO OPENSTACK.

Primero instalaremos los repositorios de openstack en todos los servidores:

```
apt-get install python-software-properties software-properties-common
add-apt-repository cloud-archive:juno
apt-get update && apt-get upgrade
```

En la instalación del previo de openstack además de instalar los paquetes necesarios openstack, tendremos que instalar también 3 distintos productos que iremos contando en los siguientes puntos, estos serán la instalación de un software de base de datos en nuestro caso será MariaDB, un software de paso de mensajes, en nuestro caso será el de RabbitMq y para que no haya una diferencia de horas instalaremos un servidor de hora para sincronizar entre si los servidores.

Lo primero que vamos a contar es como configurar el servidor de hora y los clientes a través de NTP.

3.2.1. NTP.

Network Time Protocol es el método más común para sincronizar el reloj del software de un sistema GNU/Linux con los servidores horarios de Internet. Está diseñado para mitigar los efectos de la variable de latencia de la red y, por lo general, pueden mantener el horario dentro de un margen de decenas de milisegundos respecto a Internet. La precisión en las redes de área local es aún mejor, hasta un milisegundo.

El proyecto NTP proporciona una implementación de referencia del protocolo llamado simplemente NTP. Una alternativa a NTP es Chrony, un dial-up amigable y diseñado específicamente para los sistemas que no están en línea todo el tiempo, y OpenNTPD, que forma parte del proyecto OpenBSD y que actualmente no se mantiene para Linux.

3.2.1.1. Instalación NTP.

Debido a la comunicación entre las máquinas del laboratorio de openstack es necesario que tengan la misma hora, por esta razón deberemos instalar y configurar la hora de las máquinas a través de NTP.

Configuración del servidor NTP en la máquina supervisor

```
sed -i 's/server ntp.ubuntu.com/server ntp.ubuntu.com\nserver 127.127.1.0\nfudge
127.127.1.0 stratum 10/g' /etc/ntp.conf
service ntp restart
```

Sincronizaremos las máquinas virtualizador01, virtualizador02 y red

```
service ntp stop
ntpdate 150.200.180.100
```

Configuraremos el servicio NTP en virtualizador01, virtualizador02 y red

```
vi /etc/ntp.conf
server 150.200.180.100
restrict 127.0.0.1
restrict ::1

service ntp restart
```

Una vez instalado el ntp, tendremos que instalar un servidor de base de datos para gestionar el openstack, en nuestro caso al ser un laboratorio de enseñanza hemos decidido elegir uno gratuito como MariaDB, además en el siguiente punto explicaremos las características de esta base de datos.

3.2.2. MARIADB:

MariaDB es un sistema de gestión de bases de datos derivado de MySQL con licencia GPL. Introduce dos motores de almacenamiento nuevos, uno llamado Aria -que reemplaza con ventajas a MyISAM- y otro llamado XtraDB -en sustitución de InnoDB. Tiene una alta compatibilidad con MySQL ya que posee las mismas órdenes, interfaces, APIs y bibliotecas, siendo su objetivo poder cambiar un servidor por otro directamente. MariaDB es un fork directo de MySQL.

En la práctica MariaDB reemplaza directamente a la misma versión de MySQL (MySQL 5.1 -> MariaDB 5.1, MariaDB 5.2 & MariaDB 5.3 son compatibles. MySQL 5.5 -> MariaDB 5.5). Las diferencias se encuentran en estos puntos:

- *Mecanismos de almacenamiento*

Además de los mecanismos de almacenamiento estándar MyISAM, Blackhole, CSV, Memory y Archive, también se incluyen en la versión fuente y binaria de MariaDB como por ejemplo:

Aria (alternativa a MyISAM resistente a caídas)

XtraDB (reemplazo directo de InnoDB)

Cassandra, en MariaDB 10.0 (otros mecanismos no-sql se incluirán en MariaDB)

- *Facilidad de uso*

Proporciona estadísticas de índices y tabla.

Introducidas características estilo NoSQL.

Columnas dinámicas, que proporcionan al usuario columnas virtuales en las tablas.

- *Prestaciones*

El optimizador de MariaDB -que se encuentra en el núcleo de cualquier SGBD- funciona claramente más rápido con cargas complejas.

Eliminación de tablas. El acceso a tablas a través de views acelera el acceso.

- *Testeo*

Más juegos de test en la distribución.

Parches para los test.

Menos errores y alertas

Los juegos de testeo han permitido reducir los errores sin introducir nuevos.

3.2.2.1. Instalación MariaDB

Una vez explicado las prestaciones del servidor de base de datos MariaDB procederemos a su instalación en la máquina supervisor de los paquetes necesarios para su instalación:

```
apt-get install mariadb-server python-mysqldb
```

Configuraremos el acceso a MariaDB con el usuario root:

```
vi /root/.my.cnf
[client]
host = localhost
user = root
password = root

chmod 400 /root/.my.cnf
```

Procederemos a configurar el servidor de base de datos MariaDB en el fichero de configuración /etc/mysql/my.cnf, en el fichero le definiremos la IP por donde escuchará y su tipo de caracteres:

```
bind-address            = 150.200.180.100

collation-server = utf8_general_ci
init-connect='SET NAMES utf8'
character-set-server = utf8
```

Una vez modificado el fichero de configuración reiniciaremos el servicio mysql

```
service mysql restart
```

En producción securizariamos MariaDB con el siguiente comando.

```
mysql_secure_installation
```

El siguiente punto importante en la decisión de la base de la instalación de openstack es que software de paso de mensaje se va a instalar. En nuestro caso al ser un proyecto educativo cogeremos uno gratuito como es rabbitmq. Procederé a una explicación breve de rabbitmq.

3.2.3. RABBIT-MQ:

RabbitMQ es un software de negociación de mensajes de código abierto, y entra dentro de la categoría de middleware de mensajería. Implementa el estándar Advanced Message Queuing Protocol (AMQP). Está escrito en Erlang y utiliza el framework Open Telecom Platform (OTP) para construir sus capacidades de ejecución distribuida y conmutación ante errores. El código fuente está liberado bajo la licencia Mozilla Public License.

El proyecto RabbitMQ consta de diferentes partes:

- El servidor de intercambio RabbitMQ en sí mismo
- Pasarelas para los protocolos HTTP, XMPP y STOMP.
- Bibliotecas de clientes para Java y el framework .NET.
- El plugin Shovel que se encarga de copiar (replicar) mensajes desde un corredor de mensajes a otros

Advanced Message Queuing Protocol

El estándar AMQP (Advanced Message Queuing Protocol) es un protocolo de estándar abierto en la capa de aplicaciones de un sistema de comunicación. Las características que definen al protocolo AMQP son la orientación a mensajes, encolamiento ("queuing"), enrutamiento (tanto punto-a-punto como publicación-subscripción), exactitud y seguridad.

AMQP estipula el comportamiento tanto del servidor que provee los mensajes como del cliente de la mensajería hasta el punto de que las implementaciones de diferentes proveedores son verdaderamente interoperables, de la misma manera que los protocolos SMTP, HTTP, FTP y análogos han creado sistemas interoperables. AMQP es un protocolo a nivel de cable. Un protocolo a nivel de cable es una descripción del formato de los datos que son enviados a través de la red como un flujo de octetos. En consecuencia, cualquier programa que pueda crear e interpretar mensajes conforme a este formato de datos puede interoperar con cualquier otra herramienta que cumpla con este protocolo, independientemente del lenguaje de implementación.

3.2.3.1. Instalación Rabitt-mq:

Procederemos a la instalación de los paquetes necesarios del servidor de RabbitMQ en la máquina supervisor:

apt-get install rabbitmq-server

Una vez instalado el software crearemos el fichero de configuración RabbitMQ, donde definiremos por la IP que escuchara los mensajes y porque puerto.

vi /etc/rabbitmq/rabbitmq.config

[{rabbit, [{tcp_listeners, [{"150.200.180.100", 5672},{"127.0.0.1", 5672}]}]}].

Después de crear la configuración procederemos a añadir usuarios, virtualhost y permisos:

rabbitmqctl add_user openstackuser openstackpass
rabbitmqctl add_vhost /openstack
rabbitmqctl set_permissions -p /openstack openstackuser ".*" ".*" ".*"

Comprobaremos que su creación es correcta:

rabbitmqctl list_permissions -p /openstack
rabbitmqctl list_vhosts
rabbitmqctl list_users

Una vez hecho esto procederemos a reiniciar el servicio para que coja los cambios

service rabbitmq-server restart

3.2.4. Clientes Openstack.

Por último en la máquina cliente en nuestro caso ostra.it.uc3m.es instalaremos los clientes de las APIs que vamos a trabajar en openstack, estas API se comunicarán con los servicios que instalaremos de openstack, en nuestro caso Keystone, Glance, Nova, Neutron y Cinder. A través de estos clientes podremos dar instrucciones y consultar los servicios que instalaremos.

apt-get install python-keystoneclient python-glanceclient python-novaclient python-neutronclient python-cinderclient

3.3. INSTALACIÓN OPENSTACK.

Con el laboratorio ya montado procederemos a la instalación de openstack. Este proceso se realizara montando los distintos módulos necesarios para el laboratorio.

3.3.1. Keystone.

Keystone permite la integración de los servicios de OpenStack en un único punto en aspectos tan importantes como proporcionar servicios de autenticación, gestión de tokens y el mantenimiento de un catálogo y un repositorio de políticas de identidad.

Cada función de Keystone puede conectarse a un backend distinto que permite realizar esa misma función de diferentes formas utilizando un servicio distinto. De esta forma, Keystone puede integrarse fácilmente con diferentes almacenamientos como SQL, LDAP o KVS (Key Value Stores).

Esto es muy útil en cuanto a la integración de los servicios de autenticación de OpenStack con los servicios de autenticación existentes en un despliegue en concreto.

Este módulo es el encargado del sistema de autentificación y autorización de los distintos componentes desde la versión Essex y tiene dos funciones principales:

• Gestión de usuarios: Keystone es el encargado de mantener un registro de usuarios y los permisos que tienen cada uno de ellos.

• Registro los servicios ofrecidos: Keystone ofrece un catálogo de los servicios ofrecidos, así como la forma de acceder a sus APIs.

Los conceptos fundamentales de la gestión de usuarios son:

• Usuario: Podemos guardar su nombre, correo electrónico y contraseña.

• Proyecto (tenant en la jerga de OpenStack): En un proyecto podemos ejecutar un conjunto de instancias con características en común, por ejemplo pueden estar todas las instancias en el misma red, pueden utilizar una serie de imágenes de sistemas o tener limitado el uso de recursos del cloud.

• Rol: Nos indica qué operaciones puede realizar cada usuario. A un usuario se le pueden asignar diferentes roles en cada proyecto.

Los conceptos fundamentales del registro de servicio son:

• Servicio: Corresponde a un componente de OpenStack que puede utilizar el módulo de autentificación.

• Endpoints: Representa las URL que nos permiten acceder a las API de cada uno de los servicios o componentes de OpenStack

3.3.1.1. Instalación Keystone

Una vez explicado que es keystone procederemos a instalarlo en el servidor Supervisor.

apt-get install keystone

El siguiente paso será la creación de su base de datos en el servidor MariaDB.

CREATE DATABASE keystone;
GRANT ALL ON keystone. TO 'keystoneuser'@'localhost' IDENTIFIED BY 'keystonepass';*

Debido a que keystone es un gestor de claves para poder generar las originales deberemos configurarlo con una cadena de conexión de administrador en código. Esta cadena de conexión cuando ya hayamos generado nuestro usuario procederemos a eliminarla por motivos de seguridad.

Por esta razón modificaremos el fichero de configuración del servicio keystone (/etc/keystone/keystone.conf) con los siguientes parámetros:

admin_token = ADMIN123456789TOKEN123456789

public_bind_host = 0.0.0.0
admin_bind_host = 0.0.0.0
admin_port = 35357
public_port = 5000

[database]
connection = mysql://keystoneuser:keystonepass@localhost/keystone

[token]
provider = keystone.token.providers.uuid.Provider
driver = keystone.token.persistence.backends.sql.Token

Después de cambiar la configuración deberemos reiniciar el servicio para que coja los cambios:

service keystone restart

El siguiente paso es inicializar la base de datos con el siguiente comando de keystone.

keystone-manage db_sync

Debido a que no utilizaremos sqlite sino MariaDB borraremos el fichero que nos crea por defecto ubuntu al instalar el servicio.

rm -f /var/lib/keystone/keystone.db

Para poder conectarnos a keystone en la máquina anfitrión (ostra.it.uc3m.es) generaremos un fichero con los datos de conexión (/root/openstack):

export OS_SERVICE_TOKEN=ADMIN123456789TOKEN123456789
export OS_SERVICE_ENDPOINT=http://150.200.190.100:35357/v2.0/

Una vez generado procederemos a cargarlos:

source openstackrc

Con esta variable ya cargada procederemos a crear el servicio de keystone desde la maquina anfitriona a través de la API de keystone:

keystone service-create --name=keystone --type=identity --description="Identity Service"

De esta ejecución necesitaremos guardar el KEYSTONE_SERVICE_ID que nos devuelve el comando ejecutado porque lo necesitaremos para cargar los datos de keystone en la siguiente ejecución:

keystone endpoint-create --region Madrid --service-id=<KEYSTONE_SERVICE_ID> --
publicurl=http://150.200.190.100:5000/v2.0 --
internalurl=http://150.200.180.100:5000/v2.0 --
adminurl=http://150.200.180.100:35357/v2.0

3.3.1.2. Creación de Proyectos y usuarios.

El siguiente paso es crear los tenants (proyectos) que vamos a tener, se pueden tener los tenants que queramos, necesitaremos uno por cada cliente, en nuestro caso como es para un laboratorio solo crearemos uno. En un caso real tendremos que crear uno por cliente porque a través del proyecto el cliente gestionará sus máquinas, redes,.. y solo tendrá acceso a las suyas.

keystone tenant-create --name caostack
keystone tenant-create --name service

Comprobaremos que hemos creado los dos bien:

```
root@ostra:~# keystone tenant-list
+----------------------------------+----------+---------+
|                id                |   name   | enabled |
+----------------------------------+----------+---------+
| b6edd2b7841041d8a56efac9972d28a4 | caostack |   True  |
| 7bb3e758c11046e5b0ddfb7fd52fae4d | service  |   True  |
+----------------------------------+----------+---------+
```

El siguiente paso será crear los roles de los usuarios, para el laboratório solo creamos dos, uno de administración y otro de usuario, como en el caso del tenant en un caso real se pensaría como estructurarlo, aquí con estos roles nos vale:

keystone role-create --name admin
keystone role-create --name Member

Comprobaremos que hemos creado los dos bien con:

```
root@ostra:~# keystone role-list
+----------------------------------+--------+
|                id                |  name  |
+----------------------------------+--------+
| a226a28d362a4a86995f666c867357c1 | Member |
| d0f57046f57a4951a5e9a295aa632e9c | admin  |
+----------------------------------+--------+
```

Y por último crearemos los usuarios de openstack, en nuestro caso solo creamos un usuario para administrar, ya que en nuestro caso el administrador del openstack y el que genera el proyecto somos nosotros, así que no necesitamos más:

keystone user-create --name admin --pass s3cr3t0 --email tio_xexu@yahoo.es

Comprobaremos que hemos creado el usuario:

```
root@ostra:~# keystone user-list
+----------------------------------+---------+---------+-------------------+
|                id                |   name  | enabled |       email       |
+----------------------------------+---------+---------+-------------------+
| 97f4fef867e44fd28333e1046c901002 |  admin  |   True  | tio_xexu@yahoo.es |
```

Y por último le asignaremos un rol y un tenant al usuario creado, al darle el rol admin, no necesitará el rol de Member, ya que este rol es el que tiene todos los permisos.

keystone user-role-add --tenant caostack --user admin --role admin

Después de haber creado ya el usuario procederemos a eliminar la cadena de conexión, y así eliminaremos los problemas de seguridad.

1º Eliminamos las claves cargadas:

unset OS_SERVICE_ENDPOINT
unset OS_SERVICE_TOKEN

2º Modificaremos el fichero con la configuración que creamos antes /root/openstack (aconsejo que este fichero se cargue cada vez que hagas login con el usuario):

#export OS_SERVICE_TOKEN=ADMIN123456789TOKEN123456789
#export OS_SERVICE_ENDPOINT=http://172.16.0.100:35357/v2.0/
export OS_USERNAME=admin
export OS_PASSWORD=s3cr3t0
export OS_TENANT_NAME=caostack
export OS_AUTH_URL=http://150.200.190.100:35357/v2.0

3ª Cargaremos las variables y comprobaremos que da los mismos resultados:

source openstackrc

keystone user-list

4º Viendo que ya está todo ok procederemos a eliminar la cadena de conexión en el fichero /etc/keystone/keystone.conf de la máquina supervisor.

#admin_token = ADMIN123456789TOKEN123456789

5º Reiniciamos el servicio para que aplique los cambios.

service keystone restart

3.3.2. Glance.

El proyecto Glance proporciona los servicios necesarios para la búsqueda, localización y obtención de imágenes para las máquinas virtuales del cloud. Al igual que el resto de componentes de OpenStack, Glance posee una API RESTful que permite solicitudes tanto de los metadatos de las imágenes para las máquinas virtuales, como la solicitud en sí de una imagen.

Las imágenes que están disponibles a través de Glance, se pueden almacenar en diferentes ubicaciones, desde un simple sistema de ficheros que es la opción por defecto a un sistema de almacenamiento de objetos como OpenStack Swift.

Posiblemente glance sea el componente más sencillo de todo el conjunto de proyectos de OpenStack y el ritmo de desarrollo y cambios no tiene nada que ver con el de otros componentes.

La arquitectura de Glance se ha mantenido relativamente estable desde la versión Cactus de OpenStack. El mayor cambio lo representa la incorporación de Keystone como sistema de autenticación y autorización, la cual se añadió en la versión Diablo. Glance está formado por cuatro componentes principales:

• El demonio glance-api. Encargado de aceptar peticiones a través de su API para el descubrimiento, recuperación y almacenamiento de imágenes.

• glance-registry. Encargado de almacenar, procesar y recuperar meta información sobre las imágenes (tamaño, tipo, etc.).

• Una base de datos para almacenar dicha meta información. De la misma forma que Nova, la base de datos es optativa, pero la decisión siempre gira en torno a MySQL o PostgreSQL para entornos en producción.

• Un repositorio de almacenamiento para los ficheros de imágenes. Este almacenamiento es configurable y pueden utilizarse desde directorios locales al propio servicio Swift. Otras soluciones pasan por volúmenes iSCSI, directorios NFS o CIFS, RADOS block device, Amazon S3 o HTTP.

Existen también un conjunto de servicios para la gestión de la caché.

Glance representa un papel central en la arquitectura de OpenStack, ya que acepta peticiones para la gestión de imágenes tanto de los usuarios finales como de otros servicios como Nova.

3.3.2.1. Instalación glance.

Primero lo que haremos es instalar el glance en la máquina supervisor.

apt-get install glance

Crearemos la base de datos de este servicio en el servidor MariaDB.

CREATE DATABASE glance;
GRANT ALL ON glance. TO 'glanceuser'@'localhost' IDENTIFIED BY 'glancepass';*

Configuraremos los datos de base de datos en los ficheros de configuración /etc/glance/glance-api.conf y /etc/glance/glance-registry.conf.

[database]
#sqlite_db = /var/lib/glance/glance.sqlite
connection = mysql://glanceuser:glancepass@localhost/glance

Una vez modificado reiniciaremos los servicios:

service glance-api restart
service glance-registry restart

Cuando haya cogido los cambios inicializaremos la base de datos:

glance-manage db_sync

3.3.2.2. Creación servicios y usuarios.

Una vez ya creada la base de datos e instalado el glance procederemos a instalar todo lo necesario para que glance se comunique con todos los servicios a través de Keystone.

Primero procederemos a crear los servicios desde la máquina ostra.it.uc3m.es.

keystone service-create --name=glance --type=image --description="Image Service"

De esta ejecución necesitaremos guardar el GLANCE_SERVICE_ID porque lo necesitaremos para la definirle porque url nos va a gestionar keystone las ordenes cuando le definimos su endpoint.

keystone endpoint-create --region Madrid --service-id=<GLANCE_SERVICE_ID> --publicurl=http://150.200.190.100:9292 --internalurl=http://150.200.180.100:9292 --adminurl=http://150.200.180.100:9292

Ya creados los servicios crearemos los usuarios correspondiente para gestionar el servicio.

keystone user-create --name glance --pass gl4nc3 --email tio_xexu@yahoo.es

Comprobaremos que se ha creado bien

```
root@ostra:~# keystone user-list
+----------------------------------+---------+---------+-------------------+
|                id                |   name  | enabled |       email       |
+----------------------------------+---------+---------+-------------------+
| 97f4fef867e44fd28333e1046c901002 |  admin  |   True  | tio_xexu@yahoo.es |
| 119e8e0d44594bfbb5a4958de35ff9c3 |  glance |   True  | tio_xexu@yahoo.es |
```

Y le añaderemos el rol de admin.

keystone user-role-add --tenant service --user glance --role admin

3.3.2.3.Configuración Glance.

Ya creado los servicios y los usuarios procederemos a configurar glance.

1º Configuración de la autenticación en ambos servicios.

Procederemos a configurar los datos generados por keystone (usuario, password, url) en /etc/glance/glance-api.conf

```
[keystone_authtoken]
auth_uri = http://150.200.180.100:5000/v2.0
identity_uri = http://150.200.180.100:35357
admin_tenant_name = service
admin_user = glance
admin_password = gl4nc3

[paste_deploy]
config_file = /etc/glance/glance-api-paste.ini
flavor=keystone
```

También procederemos a configurar los datos generados por keystone (usuario, password, url) en /etc/glance/glance-registry.conf

```
[keystone_authtoken]
auth_uri = http://150.200.180.100:5000/v2.0
identity_uri = http://150.200.180.100:35357
admin_tenant_name = service
admin_user = glance
admin_password = gl4nc3


[paste_deploy]
config_file = /etc/glance/glance-registry-paste.ini
flavor=keystone
```

2º Configuración del almacenamiento

Procederemos a definir donde y con qué formato vamos a guardar las imágenes de sistema operativo que iremos a guardar, esto lo haremos en el fichero de configuración /etc/glance/glance-api.conf

```
default_store = file

filesystem_store_datadir = /var/lib/glance/images/
```

3º Configuración de las apis a usar.

En nuestro caso hemos decidido utilizar las dos api que tiene Juno en el futuro con configurar la v2 nos valdría pero por compatibilidad hacia atrás configuraremos las dos que no nos hace ningún mal en el fichero /etc/glance/glance-api.conf

enable_v1_api=True
enable_v2_api=True

4º Configuración de IP y puerto en el servicio.

Además deberemos configurar por donde va a dar servicio (Ip y puerto) la api de glance en el fichero /etc/glance/glance-api.conf

bind_host = 0.0.0.0
bind_port = 9292

registry_host = 127.0.0.1
registry_port = 9191

Acabamos configurando los datos necesarios para el servicio de registry dándole su Ip y puerto necesarios en /etc/glance/glance-registry.conf

bind_host = 127.0.0.1
bind_port = 9191

Como vemos después de ver la configuración registry escucha solo en local en el servidor Supervisor y la api por el contrario escucha por todos los puertos para poder comunicarse con el resto de servicios de openstack.

5º Reinicio ambos servicios.

service glance-api restart
service glance-registry restart

Para nuestro futuro laboratorio procederemos a configurar un par de imagines con la que trabajaremos más adelante y así comprobaremos el buen funcionamiento del servicio de glance.

Primero crearemos una imagen desde un fichero local, lo que haremos es bajarla de internet:

wget http://cdn.download.cirros-cloud.net/0.3.3/cirros-0.3.3-x86_64-disk.img

El siguiente paso es introducirlo en glance, para que podamos utilizarla en openstack cuando la necesitemos.

glance image-create --name="Cirros 0.3.3" --disk-format qcow2 --container-format bare --is-public true < cirros-0.3.3-x86_64-disk.img

En el caso anterior creamos una imagen a través de un fichero que guardamos nosotros (esto es lo más óptimo), pero si tuviéramos problemas de almacenamiento y no quisiéramos bajarla podemos crear una imagen a partir de una URL en local

glance image-create --name="Cirros 0.3.3 (URL)" --disk-format qcow2 --container-format bare --is-public true --location http://download.cirros-cloud.net/0.3.3/cirros-0.3.3-x86_64-disk.img

Para comprobar que se han cargado bien ejecutamos:

```
root@ostra:~# glance image-list
+--------------------------------------+--------------------+-------------+------------------+----------+--------+
| ID                                   | Name               | Disk Format | Container Format | Size     | Status |
+--------------------------------------+--------------------+-------------+------------------+----------+--------+
| cb2ced70-2172-489b-b5c2-0b9c9a7195cd | Cirros 0.3.3       | qcow2       | bare             | 13200896 | active |
| a31ac549-fe9e-42c4-998c-039ececa760f | Cirros 0.3.3 (URL) | qcow2       | bare             | 13200896 | active |
+--------------------------------------+--------------------+-------------+------------------+----------+--------+
```

3.3.3. Nova

Nova no ha cambiado mucho desde las anteriores versiones, se han añadido ciertas mejoras en determinados servicios para la compatibilidad de EC2 y servicios de consola.

Nova depende de los siguientes demonios para su funcionamiento:

- **nova-api** es la encargada de aceptar y responder a las llamadas del usuario final a las APIs de nova-compute y nova-volume. El demonio nova-api soporta la API de OpenStack, la API EC2 de Amazon y la API especial de administración (para usuarios con privilegios que realicen tareas administrativas).

Además, este demonio es el encargado de la coordinación de ciertas actividades (como la ejecución de una instancia) y la aplicación de ciertas políticas (como la comprobación de cuotas).

En Essex, nova-api se ha modularizado, permitiendo únicamente la ejecución de determinadas APIs.

- El demonio **nova-compute** es el principal encargado de crear y acabar con las máquinas virtuales (instancias) utilizando para ello las APIs del hipervisor utilizado. nova-compute utiliza libvirt para KVM/QEMU, XenAPI para XenServer/XCP y VMwareAPI para Vmware.

El proceso completo de creación/destrucción de instancias es bastante complejo, pero la base es muy simple: aceptar acciones de la cola de mensajes y ejecutar un conjunto de comandos del sistema asociados (como lanzar una instancia de KVM), todo mientras que se actualiza el estado en la base de datos.

- **nova-volume** gestiona la creación, conexión y desconexión de volúmenes persistentes a las instancias, de forma similar a como lo realizar el servicio EBS (Elastic Block Storage) de Amazon. Se pueden utilizar volúmenes de diferentes proveedores como iSCSI o RADOS Block Device (RBD) de Ceph .

- El demonio **nova-network** es muy parecido a los demonios nova-compute y nova-volume. Acepta tareas de red desde la cola de mensajes y realiza ciertas que modifican el estado de la red, como por ejemplo configurar una interfaz bridge o cambiar las reglas de iptables.

El demonio **nova-scheduler** es conceptualmente la pieza de código más simple de Nova. A partir de un mensaje de solicitud de creación de una instancia, determina qué nodo de OpenStack debe ejecutar dicha instancia de acuerdo a un algoritmo previamente seleccionado. La elección se realiza entre todos los nodos que ejecutan el demonio nova-compute.

• La cola de mensajes (queue) proporciona un hub centralizado para el intercambio de mensajes entre todos los demonios. Como cola de mensajes, se utiliza actualmente RabbitMQ, pero se puede utilizar cualquier otra cola de mensajes compatible con AMQP como por ejemplo Apache Qpid.

• Una base de datos SQL. El sistema gestor de BBDD será el encargado de almacenar toda la información del cloud así como el estado inicial y de ejecución. Esto incluye los tipos de instancia que están disponibles para el uso, las instancias creadas en un momento determinado, las redes disponibles, los proyectos existentes, etc. Teóricamente, Nova soporta cualquier base de datos soportada por SQL-Alchemy, pero las bases de datos realmente utilizadas actualmente son PostgreSQL, MySQL y sqlite3 (esta última solo para pruebas y desarrollo).

Durante las últimas revisiones, Nova ha visto aumentado sus servicios de consola. Los servicios de consola permiten a los usuarios finales acceder a sus máquinas virtuales a través de una consola de texto (caso de GNU/Linux) o consola gráfica (caso de máquinas virtuales Linux y Windows). Este acceso se realiza utilizando un proxy y se basa en los demonios **nova-console** y **nova-consoleauth**

Nova interactúa con el resto de servicios de la forma esperada: con Keystone para la autenticación, con Glance para la recuperación de imágenes y con Horizon para la interfaz web. La interacción con Glance es interesante, El proceso nova-api puede subir imágenes y consultar a Glance, mientras que nova-compute se descargará la imagen necesaria para lanzar una nueva instancia.

3.3.3.1. Instalación nova.

Primer paso es la instalación de nova, instalamos el software en la máquina supervisor, en nuestro caso nova-network y nova-volume no serán necesarios ya que instalaremos los servicios de neutron y cinder que los sustituyen en versiones más modernas.

apt-get install nova-api nova-conductor nova-scheduler nova-novncproxy nova-cert nova-consoleauth

Luego tendremos que instalar el modulo nova-compute en las máquinas donde vamos a crear las máquinas virtuales, en nuestro caso en virtualizador01 y virtualizador02

apt-get install nova-compute sysfsutils

Ahora procederemos a la configuración de nova en la máquina supervisor que es la que va a gestionar el servicio. Para ello realizaremos los siguientes pasos:

1º *Borrado de la base de datos sqlite*

rm /var/lib/nova/nova.sqlite

2º *Creamos la base de datos en el mariaDB*

CREATE DATABASE nova;
GRANT ALL ON nova. TO 'novauser'@'localhost' IDENTIFIED BY 'novapass';*

3º *Configuramos nova para que se conecte a MariaDB en el fichero /etc/nova/nova.conf*

[database]
connection = mysql://novauser:novapass@localhost/nova

4º *Reiniciamos el servicio de nova en supervisor.*

service nova-api restart

5º *Inicializamos la base de datos.*

nova-manage db sync

3.3.3.2. Creación de servicio y usuario.

Una vez ya creada la base de datos e instalado el nova procederemos a crear los servicios desde la máquina ostra.it.uc3m.es

keystone service-create --name=nova --type=compute --description="Compute Service"

De esta ejecución necesitaremos guardar el NOVA_SERVICE_ID porque lo necesitaremos para la definirle porque url nos va a gestionar keystone las órdenes cuando le definimos su endpoin:

keystone endpoint-create --region Madrid --service-id=<NOVA_SERVICE_ID> --publicurl="http://150.200.190.100:8774/v2/%(tenant_id)s" --internalurl="http://150.200.180.100:8774/v2/%(tenant_id)s" --adminurl="http://150.200.180.100:8774/v2/%(tenant_id)s"

Ya creados los servicios crearemos el usuario correspondiente para poder gestionar el servicio de nova.

keystone user-create --name nova --pass n0v4 --email tio_xexu@yahoo.es

Comprobaremos que se ha creado bien

```
root@ostra:~# keystone user-list
+----------------------------------+---------+---------+-------------------+
|                id                |   name  | enabled |       email       |
+----------------------------------+---------+---------+-------------------+
| 97f4fef867e44fd28333e1046c901002 |  admin  |   True  | tio_xexu@yahoo.es |
| 119e8e0d44594bfbb5a4958de35ff9c3 |  glance |   True  | tio_xexu@yahoo.es |
| 47ed1c8fb05c48039ab71313fe0a6097 |   nova  |   True  | tio_xexu@yahoo.es |
+----------------------------------+---------+---------+-------------------+
```

Y le añadiremos el rol de admin.

keystone user-role-add --tenant service --user nova --role admin

3.3.3.3.Configuración Nova.

Después de crear los servicios procederemos a configurar nova.

1º *Configuración de la autenticación*

En el fichero /etc/nova/nova.conf en las máquinas supervisor, virtualizador01 y vituralizador02 procederemos a introducir los datos generados en el punto anterior con Keystone.

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
auth_uri = http://150.200.180.100:5000/v2.0
identity_uri = http://150.200.180.100:35357
admin_tenant_name = service
admin_user = nova
admin_password = n0v4
```

2º *Configuración de la IP de gestión*

En el fichero de configuración de nova /etc/nova/nova.conf de la máquina donde reside nova (la máquina supervisor).

```
[DEFAULT]
...
my_ip = 150.200.180.100
```

También tenemos que configurar la ip y el puerto por donde dará servicio nova-api, esto se define en el fichero /etc/nova/nova.conf de la máquina donde reside nova (la máquina supervisor).

```
[DEFAULT]
...
osapi_compute_listen = 0.0.0.0
osapi_compute_listen_host = 8774
```

3º *Configuración de los servicios*

Tendremos que definir donde se encuentran los directorios y ficheros de configuración en /etc/nova/nova.conf en las 3 máquinas, en nuestro caso definiremos los de por defecto de la instalación:

[DEFAULT]
...
verbose = True
logdir = /var/log/nova
state_path = /var/lib/nova
lock_path = /var/lock/nova
rootwrap_config = /etc/nova/rootwrap.conf
api_paste_config = /etc/nova/api-paste.ini

4º *Configuración el rabittmq*

Tendremos que definir que gestor de colas utilizamos (en nuestro caso rabbit-mq), donde se encuentra (ip y puerto) y con qué usuario en el fichero de configuración /etc/nova/nova.conf en las 3 máquinas:

[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = 150.200.180.100
rabbit_port = 5672
rabbit_use_ssl = false
rabbit_userid = openstackuser
rabbit_password = openstackpass
rabbit_virtual_host = /openstack

5º *Configuración de nova-compute*

Configuraremos nova-compute en las máquinas Virtualizador01 y Virtualizador02 en el fichero /etc/nova/nova.conf, en nuestro caso le diremos que virtualizador vamos a utilizar, como explicamos anteriormente nosotros hemos decidido usar kvm.

[DEFAULT]
...
compute_driver=libvirt.LibvirtDriver
libvirt_type = kvm

6º *Configuración de glance.*

Configuraremos glance a través de nova en el fichero /etc/nova/nova.conf de la máquina Supervisor:

[glance]
host=150.200.180.100

7º *Configuración de nova-novncproxy.*

En la máquina supervisor tendremos que definirle en el fichero /etc/nova/nova.conf por la url que escucha el servicio.

[DEFAULT]
...
novncproxy_base_url=http://150.200.190.100:6080/vnc_auto.html

En las máquinas Virtualizador01 y Virtualizador02 tendremos que definirle en el fichero /etc/nova.conf por la url que escucha el servicio como en la máquina supervisor, pero también tenemos que definirle la configuración del cliente, en la definición la x cambia si es Virtualizador01 (1) o Virtualizador02 (2):

[DEFAULT]
...
novncproxy_base_url=http://150.200.190.100:6080/vnc_auto.html
vncserver_proxyclient_address=150.200.180.10x
vncserver_listen=150.200.180.10x
vnc_keymap=es

8º *Configuración de zona.*

Definiremos una zona de disponibilidad por defecto en la máquina supervisor en /etc/nova/nova.conf, esto lo utilizaremos en el futuro con cinder, pero lo definimos ahora y lo veremos en la parte de cinder:

[DEFAULT]
...
default_availability_zone = zone01

9º *Reinicio de servicios.*

Reiniciaremos todos los servicios en las 3 máquinas.

Primero reiniciamos los servicios en supervisor:

service nova-api restart
service nova-conductor restart
service nova-cert restart
service nova-consoleauth restart
service nova-novncproxy restart
service nova-scheduler restart

Para terminar reiniciamos nova-compute en virtualizador01 y virtualizador02

service nova-compute restart

Podemos comprobar que los servicios están corriendo de dos formas distintas:

- Desde la máquina supervisor:

```
root@supervisor:~# nova-manage service list
Binary           Host                                 Zone             Status     State Updated_At
nova-conductor   supervisor                           internal         enabled    :-)   2015-07-01 16:25:15
nova-cert        supervisor                           internal         enabled    :-)   2015-07-01 16:25:19
nova-consoleauth supervisor                           internal         enabled    :-)   2015-07-01 16:25:19
nova-scheduler   supervisor                           internal         enabled    :-)   2015-07-01 16:25:16
nova-compute     virtualizador01                      zone01           enabled    :-)   2015-07-01 16:25:12
nova-compute     virtualizador02                      zone01           enabled    :-)   2015-07-01 16:25:15
```

- Desde la máquina local ostra.it.uc3m.es a través de la API.

```
root@ostra:~# nova service-list
+----+------------------+----------------+----------+---------+-------+----------------------------+-----------------+
| Id | Binary           | Host           | Zone     | Status  | State | Updated_at                 | Disabled Reason |
+----+------------------+----------------+----------+---------+-------+----------------------------+-----------------+
| 1  | nova-conductor   | supervisor     | internal | enabled | up    | 2015-07-01T16:27:45.000000 | -               |
| 2  | nova-cert        | supervisor     | internal | enabled | up    | 2015-07-01T16:27:49.000000 | -               |
| 3  | nova-consoleauth | supervisor     | internal | enabled | up    | 2015-07-01T16:27:49.000000 | -               |
| 4  | nova-scheduler   | supervisor     | internal | enabled | up    | 2015-07-01T16:27:46.000000 | -               |
| 5  | nova-compute     | virtualizador01 | zone01   | enabled | up    | 2015-07-01T16:27:52.000000 | -               |
| 6  | nova-compute     | virtualizador02 | zone01   | enabled | up    | 2015-07-01T16:27:45.000000 | -               |
+----+------------------+----------------+----------+---------+-------+----------------------------+-----------------+
```

3.3.4. Neutrón.

OpenStack neutrón es un proyecto de red SDN que se focaliza en ofrecer redes como servicio (NaaS) en entornos computacionales virtuales. Neutrón ha reemplazado el original red programa interfaz de aplicaciones (API), llamada Quantum, en OpenStack. Neutrón está diseñado para solucionar las carencias en tecnología de redes "baked-in" encontradas en entornos cloud, así como la falta de control de tenant en entornos multiusuario sobre la topología de red y direccionamiento, lo cual resulta difícil de implementar servicios de redes avanzados.

Neutrón es el componente que permite la virtualización de red en OpenStack.

OpenStack neutrón proporciona una manera para aliviar la tensión en la red que las organizaciones en entornos cloud para hacerlo más fácil entregar NaaS en la nube. Está diseñado para proveer un mecanismo de "plug-in" que proporcionará una opción para que los operadores de red permitan diferentes tecnologías mediante la API de Quantum. También permite los inquilinos crear múltiples redes privadas y controlar la IP abordarlos. Como resultado de las extensiones de API, las organizaciones tienen control adicional sobre seguridad y políticas de cumplimiento de normas, calidad de servicio [QoS], supervisión y resolución de problemas, así como la habilidad para desplegar fácilmente servicios de red avanzadas, como un cortafuego, detección de intrusiones o VPN.

Neutrón (conocido entonces como Quantum) fue incorporado al proyecto principal durante el lanzamiento de Folsom. La comunidad OpenStack Networking está trabajando activamente en las mejoras de Neutron.

3.3.4.1. Creación base de datos, servicio y usuario

Como en todos los servicios tendremos que crear la base de datos en la máquina supervisor en el MariaDB.

```
CREATE DATABASE neutron;
GRANT ALL PRIVILEGES ON neutron.* TO neutronuser@localhost IDENTIFIED BY
'neutronpass';
```

Luego crearemos los servicios en la máquina local ostra.it.uc3m.es

```
keystone service-create --name=neutron --type=network --description="Networking
Service"
```

De esta ejecución necesitaremos guardar el NEUTRON_SERVICE_ID porque lo necesitaremos para la definirle porque url nos va a gestionar keystone las órdenes cuando le definimos su endpoint:

```
keystone endpoint-create --region Madrid --service-id=<NEUTRON_SERVICE_ID> --
publicurl=http://150.200.190.100:9696 --internalurl=http://150.200.180.100:9696 --
adminurl=http://150.200.180.100:9696
```

Ya creados los servicios crearemos los usuarios correspondientes.

```
keystone user-create --name neutron --pass n37tr0n --email tio_xexu@yahoo.es
```

Comprobaremos que se ha creado bien

```
root@ostra:~# keystone user-list
+----------------------------------+---------+---------+-------------------+
|                id                |   name  | enabled |       email       |
+----------------------------------+---------+---------+-------------------+
| 97f4fef867e44fd28333e1046c901002 |  admin  |   True  | tio_xexu@yahoo.es |
| 119e8e0d44594bfbb5a4958de35ff9c3 |  glance |   True  | tio_xexu@yahoo.es |
| 8c72d9f81b534329aeeb371523b2d1c6 | neutron |   True  | tio_xexu@yahoo.es |
| 47ed1c8fb05c48039ab71313fe0a6097 |   nova  |   True  | tio_xexu@yahoo.es |
+----------------------------------+---------+---------+-------------------+
```

Y le añadiremos el rol de admin.

```
keystone user-role-add --tenant service --user neutron --role admin
```

3.3.4.2. Configuración de Neutron y ML2.

Ahora tendremos que configurar todas las máquinas del proyecto este servicio necesita configuración en todas y por ello vamos a explicarlo en tres grupos de máquinas:

3.3.4.2.1. Configuración en la máquina Supervisor.

Instalamos los paquetes necesarios para la máquina supervisor:

```
apt-get install neutron-server neutron-plugin-ml2
```

A partir de aquí modificamos el fichero de configuración de neutron /etc/neutron/neutron.conf para definir los valores necesarios para la ejecución de neutron-server en la máquina supervisor:

- Metemos la cadena de conexión a la base de datos.

```
[database]
connection = mysql://neutronuser:neutronpass@localhost/neutron
```

- Configuramos el método de autenticación y definimos sus valores de conexión para su funcionamiento:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
auth_uri = http://150.200.180.100:5000/v2.0
identity_uri = http://150.200.180.100:35357
admin_tenant_name = service
admin_user = neutron
admin_password = n37tr0n
```

- Añadimos en la configuración por qué Ip y por qué puerto escuchara el servicio neutron-server:

```
bind_host = 0.0.0.0
bind_port = 9696
```

- Definiremos que gestor de colas utilizamos (en nuestro caso rabbit-mq), donde se encuentra (ip y puerto) y con que usuario en el fichero de configuración.

rpc_backend = rabbit
rabbit_host = 150.200.180.100
rabbit_userid = openstackuser
rabbit_password = openstackpass
rabbit_virtual_host = /openstack

- Configuramos la comunicación entre neutrón y nova, para ello tendremos que definirle por qué url escucha Nova y los datos de conexión con Nova (usuario, password...), el TENANT_SERVICE_ID lo sacaremos del punto de instalación de nova o a través del Api de nova consultando el servicio.

notify_nova_on_port_status_changes = True
notify_nova_on_port_data_changes = True
nova_url = http://150.200.180.100:8774/v2
nova_region_name = Madrid
nova_admin_username = nova
nova_admin_tenant_id = <TENANT_SERVICE_ID>
nova_admin_password = n0v4
nova_admin_auth_url = http://150.200.180.100:35357/v2.0

- Configuramos en el fichero el plugin que vamos a utilizar en nuestro caso para neutron, hemos ML2, hemos decidido este plugin por sus siguientes características:

- Puede usar múltiples tecnologías L2 de forma concurrente.
- Son tecnologías presentes en datacenters
- Actualmente funciona con agentes Open vSwitch, Linux Bridge e Hyper-V L2

Procederemos a configurarlo en el fichero /etc/neutron/neutron.conf:

core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True

- Comentamos el service_providers porque no lo vamos a utilizar:

#service_provider=LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
#service_provider=VPN:openswan:neutron.services.vpn.service_drivers.ipsec.IPsecVPNDriver:default

- Para la creación de las redes virtuales el plugin ML2 usa el agente de Open vSwitch (OVS), por lo que es necesario configurar el fichero /etc/neutron/plugins/ml2/ml2_conf.ini, existen diferentes tipos de configuración pudiendo usar Neutron con GRE (segmentación) o a través de vLASN, o incluso usando software de terceros como puede ser Vmware NSX, en nuestro caso voy a configurar a través de GRE (segmentación).

Para usar flat, el driver de red generic routing encapsulation (GRE), redes de tenant GRE y el agente de OVS lo haremos con los siguientes parámetros:

```
[ml2]
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = openvswitch
```

- Ahora indicaremos el rango de túneles que tendremos disponibles para GRE:

```
[ml2_type_gre]
tunnel_id_ranges = 1:1000
```

- Y por último habilitaremos las políticas de grupos, asignación de IP e iptable como firewall para OVS

```
[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

El último paso será configurar la parte de nova relacionada con neutrón en el fichero /etc/nova/nova.conf:

- Por defecto en Nova aun viene configurado con Nova-Network, por lo que necesitamos editar el fichero de configuración */etc/nova/nova.conf* y realizar los siguientes cambios para habilitar Neutron:

```
[DEFAULT]
...
network_api_class = nova.network.neutronv2.api.API
security_group_api = neutron
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver

[neutron]
url = http://150.200.180.100:9696
auth_strategy = keystone
admin_auth_url = http://150.200.180.100:35357/v2.0
admin_tenant_name = service
admin_username = neutron
admin_password = n37tr0n
```

- Configuración de los metadatos

```
[DEFAULT]
...
neutron_metadata_proxy_shared_secret = 1234567890
service_neutron_metadata_proxy = true
```

Procedemos a crear la base de datos de neutrón en MariaDB

```
neutron-db-manage --config-file /etc/neutron/neutron.conf \
  --config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade juno
```

Procedemos a reiniciar todos los servicios para que cojan los cambios realizados en los puntos anteriores.

```
service neutron-server restart

service nova-api restart
service nova-conductor restart
service nova-cert restart
service nova-consoleauth restart
service nova-novncproxy restart
service nova-scheduler restart
```

Para terminar desde la máquina ostra.it.uc3m.es procederemos a comprobar los módulos de neutrón.

```
root@ostra:~# neutron ext-list
+-----------------------+-----------------------------------------------+
| alias                 | name                                          |
+-----------------------+-----------------------------------------------+
| service-type          | Neutron Service Type Management               |
| ext-gw-mode           | Neutron L3 Configurable external gateway mode |
| security-group        | security-group                                |
| l3_agent_scheduler    | L3 Agent Scheduler                            |
| lbaas_agent_scheduler | Loadbalancer Agent Scheduler                  |
| fwaas                 | Firewall service                              |
| binding               | Port Binding                                  |
| provider              | Provider Network                              |
| agent                 | agent                                         |
| quotas                | Quota management support                      |
| dhcp_agent_scheduler  | DHCP Agent Scheduler                          |
| l3-ha                 | HA Router extension                           |
| multi-provider        | Multi Provider Network                        |
| external-net          | Neutron external network                      |
| router                | Neutron L3 Router                             |
| allowed-address-pairs | Allowed Address Pairs                         |
| extraroute            | Neutron Extra Route                           |
| extra_dhcp_opt        | Neutron Extra DHCP opts                       |
| lbaas                 | LoadBalancing service                         |
| dvr                   | Distributed Virtual Router                    |
+-----------------------+-----------------------------------------------+
```

3.3.4.2.2. Configuración en la máquina Red.

Instalamos los paquetes necesarios para la máquina Red, como son los de ML2, openvswitch, el de L3, dchp e Iptables.

apt-get install neutron-plugin-ml2 neutron-plugin-openvswitch-agent neutron-l3-agent neutron-dhcp-agent iptables

Configuraremos las variables del sistema operativo red en el fichero /etc/sysctl.conf

net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0

Cargaremos las nuevas variables.

sysctl -p

Ahora pasaremos a la configuración de los parámetros de neutrón en el fichero /etc/neutron/neutron.conf.

- Primero configuraremos la autenticación a través de Keystone contra el servicio neutrón con los valores generados anteriormente.

```
auth_strategy = keystone

[keystone_authtoken]
auth_uri = http://150.200.180.100:5000/v2.0
identity_uri = http://150.200.180.100:35357
admin_tenant_name = service
admin_user = neutron
admin_password = n37tr0n
```

- Definiremos que gestor de colas utilizamos (en nuestro caso rabbit-mq), donde se encuentra (ip y puerto) y con que usuario en el fichero de configuración.

```
rpc_backend = rabbit
rabbit_host = 150.200.180.100
rabbit_userid = openstackuser
rabbit_password = openstackpass
rabbit_virtual_host = /openstack
```

- Comentaremos el service_providers ya que por ahora no lo utilizaremos.

```
#service_provider=LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.hap
roxy.plugin_driver.HaproxyOnHostPluginDriver:default
#service_provider=VPN:openswan:neutron.services.vpn.service_drivers.ipsec.IPsecVPNDr
iver:default
```

- Vamos a habilitar los plugins Modular Layer 2 (ML2), servicio de router y overlapping IP addresses:

```
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

- Una vez ya realizada la configuración de neutrón deberemos configurar el ML2 en su fichero de configuración /etc/neutron/plugins/ml2/ml2_conf.inim en este caso configuraremos que utilice GRE con opensvswitch.

```
[ml2]
type_drivers = gre
tenant_network_types = gre
mechanism_drivers = openvswitch
```

- Ahora indicaremos el rango de túneles que tendremos disponibles para GRE:

```
[ml2_type_gre]
tunnel_id_ranges = 1:1000
```

- Habilitaremos las políticas de grupos, asignación de IP e iptable como firewall para OVS

```
[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

- Y por último definiremos por que IP gestionará la máquina Red y que tipo de encapsulamiento (en nuestro caso GRE), el openvswitch.

```
[ovs]
local_ip = 10.26.0.199
tunnel_type = gre
enable_tunneling = True
bridge_mappings = external:br-ex
```

- Después de meter los valores en el fichero de configuración de ML2 crearemos los bridges, en nuestro caso un bridge interno y un bridge externo.

```
service openvswitch-switch restart
ovs-vsctl add-br br-int
ovs-vsctl add-br br-ex
```

- Una vez creado los bridges incluiremos el interfaz externa en el bridge br-ex.

```
ovs-vsctl add-port br-ex eth0
```

- Para que esto funcione cambiaremos la configuración de red de la máquina red para que pueda utilizar los bridges de ml2. Deberemos tocar el fichero /etc/network/interfaces con la siguiente configuración

```
auto eth0
iface eth0 inet manual
  up ip address add 0/0 dev $IFACE
  up ip link set $IFACE up
  up ip link set $IFACE promisc on
  down ip link set $IFACE promisc off
  down ip link set $IFACE down

auto br-ex
iface br-ex inet static
  address 150.200.190.199
  netmask 255.255.255.0
  gateway 150.200.190.254
  dns-nameservers 150.200.190.254
```

Seguiremos configurando los componentes necesarios de neutrón en nuestro caso.

-El próximo en configurar es neutron-l3-agent, este componente ofrece L3/NAT fordwarding para el acceso hacia afuera de las redes. Lo haremos modificando su fichero de configuración l3_agent.ini

```
[DEFAULT]

interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces = True
external_network_bridge = br-ex
```

- El siguiente en configurar es el neutron-dhcp-agent, este plugin ofrece DHCP para las redes. Lo haremos modificando su fichero de configuración dhcp_agent.ini.

```
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
ovs_integration_bridge = br-int
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
use_namespaces = True
enable_isolated_metadata = True
dhcp_domain =
dnsmasq_dns_servers = 8.8.8.8
```

- Y por último configuraremos la metadata en el fichero metadata_agent.ini

```
auth_url = http://150.200.180.100:5000/v2.0
auth_region = Madrid
admin_tenant_name = service
admin_user = neutron
admin_password = n37tr0n

nova_metadata_ip = 150.200.180.100
nova_metadata_port = 8775

metadata_proxy_shared_secret = 1234567890
```

Ya modificados todos los ficheros de configuración de la máquina red procederemos a reiniciar sus servicios para que carguen los nuevos ficheros de configuración:

```
service neutron-dhcp-agent restart
service neutron-l3-agent restart
service neutron-metadata-agent restart
service neutron-plugin-openvswitch-agent restart
```

3.3.4.2.3 Configuración en las máquinas virtualizadoras.

Los próximos cambios que realizaremos tendrán que ser en ambas máquinas Virtualizador01 y Virtualizador02, eso si hay que tener cuidado con la IP de cada una de ellas.

Primero configuraremos el /etc/sysctl.conf y lo cargaremos.

```
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0

sysctl -p
```

Para empezar a configurar los virtualizadores deberemos instalar los paquetes de neutrón necesarios en nuestro caso el plugin de ML2 y el de openvswitch.

```
apt-get install neutron-plugin-ml2 neutron-plugin-openvswitch-agent
```

Una vez instalado procederemos a la configuración de ambas máquinas a través del fichero /etc/neutron/neutron.conf.

- Configuramos el método de autenticación y definimos sus valores de conexión para su funcionamiento:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
auth_uri = http://150.200.180.100:5000/v2.0
identity_uri = http://150.200.180.100:35357
admin_tenant_name = service
admin_user = neutron
admin_password = n37tr0n
```

- Configuraremos los dos servidores para que neutron utilice el plugin de ML2.

```
[DEFAULT]
...
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

- Definiremos que gestor de colas utilizamos (en nuestro caso rabbit-mq), donde se encuentra (ip y puerto) y con que usuario en el fichero de configuración.

```
rpc_backend = rabbit
rabbit_host = 150.200.180.100
rabbit_userid = openstackuser
rabbit_password = openstackpass
rabbit_virtual_host = /openstack
```

- Comentaremos el service_providers ya que no los vamos a utilizar:

```
#service_provider=LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
#service_provider=VPN:openswan:neutron.services.vpn.service_drivers.ipsec.IPsecVPNDriver:default
```

Una vez ya realizada la configuración de neutrón deberemos configurar el ML2 en su fichero de configuración /etc/neutron/plugins/ml2/ml2_conf.inim en este caso configuraremos que utilice GRE con opensvswitch.

```
[ml2]
type_drivers = gre
tenant_network_types = gre
mechanism_drivers = openvswitch
```

Ahora indicaremos el rango de túneles que tendremos disponibles para GRE:

```
[ml2_type_gre]
tunnel_id_ranges = 1:1000
```

Habilitaremos las políticas de grupos, asignación de IP e iptables como firewall para OVS

```
[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

Y por último definiremos por que IP local por donde correrá el ovs y que tipo de encapsulamiento (en nuestro caso GRE), el openvswitch.

```
[ovs]
local_ip = 10.26.0.10x  (la x dependiendo de la máquina)
tunnel_type = gre
enable_tunneling = True
```

Después de realizar la configuración de la parte de neutron y de los plugins que vamos a utilizar crearemos un bridge br-int.

```
service openvswitch-switch restart
ovs-vsctl add-br br-int
```

Por defecto en Nova aun viene configurado con Nova-Network, por lo que necesitamos editar el fichero de configuración */etc/nova/nova.conf* y realizar los siguientes cambios para habilitar Neutron:

```
[DEFAULT]
...
network_api_class = nova.network.neutronv2.api.API
security_group_api = neutron
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver

[neutron]
url = http://150.200.180.100:9696
auth_strategy = keystone
admin_auth_url = http://150.200.180.100:35357/v2.0
admin_tenant_name = service
admin_username = neutron
admin_password = n37tr0n
```

Para terminar reiniciaremos los servicios.

```
service neutron-plugin-openvswitch-agent restart
service nova-compute restart
```

Una vez reiniciado procederemos a hacer unas comprobaciones desde la máquina ostra.it.uc3m.es para vez que los servicios están bien:

- Listado de agentes:

```
root@ostra:~# neutron agent-list
+--------------------------------------+--------------------+----------------+-------+----------------+---------------------------+
| id                                   | agent_type         | host           | alive | admin_state_up | binary                    |
+--------------------------------------+--------------------+----------------+-------+----------------+---------------------------+
| 11a1b4ab-88af-4f58-bc54-9423b9fb2705 | DHCP agent         | red            | :-)   | True           | neutron-dhcp-agent        |
| 718f2198-fab4-4551-a8a3-e36c5ec19ab3 | L3 agent           | red            | :-)   | True           | neutron-l3-agent          |
| 7ddde76d-c156-4d1b-a79c-0503160da42c | Metadata agent     | red            | :-)   | True           | neutron-metadata-agent    |
| 890313a3-0607-4f9b-b41c-1e62361cadc1 | Loadbalancer agent | red            | :-)   | True           | neutron-lbaas-agent       |
| 8af744ce-3a84-4786-b93d-01c856300fb2 | Open vSwitch agent | virtualizador01 | :-)   | True           | neutron-openvswitch-agent |
| c6fa6abc-b315-422d-a641-edfaf71138d1 | Open vSwitch agent | red            | :-)   | True           | neutron-openvswitch-agent |
| f0816ea3-250e-4a07-bb7f-6e3bb210101c | Open vSwitch agent | virtualizador02 | :-)   | True           | neutron-openvswitch-agent |
+--------------------------------------+--------------------+----------------+-------+----------------+---------------------------+
```

- Listado de redes:

```
neutron net-list
```

- Listado de routers:

neutron router-list

En nuestro caso no habrá ni redes, ni router, porque todavía no hemos creado ninguno.

Después de comprobar que todo está bien procederemos a crear la red externa:

neutron net-create external --shared --router:external=True

Y después su subred:

neutron subnet-create external --allocation-pool start=150.200.190.200,end=150.200.190.215 --gateway=150.200.190.254 --disable-dhcp 150.200.190.0/24

3.3.4.3. Configuración servicios extras LwaaS y FwaaS.

Como extra de la configuración de neutron en openstack vamos a configurar dos servicios comunes que puede tener cualquier empresa del país, estos son FwaaS (Firewall como servicio) y LbaaS (Balanceador de carga como servicio).

1º *Configuración en la máquina Supervisor:*

En el fichero /etc/neutron/neutron.conf añadiremos, que vamos a utilizar los servicios de FwaaS y LbaaS:

```
[DEFAULT]
...
service_plugins = router,firewall,lbaas

[service_providers]
...
service_provider=LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.hapr
oxy.plugin_driver.HaproxyOnHostPluginDriver:default
```

Una vez modificado el fichero reiniciaremos el servicio del server:

```
service neutron-server restart
```

2º *Configuración de la máquina red:*

En el fichero /etc/neutron/neutron.conf añadiremos igualmente que en la máquina supervisor los servicios anteriormente dichos:

```
[DEFAULT]
...
service_plugins = router,firewall,lbaas

[service_providers]
...
service_provider=LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.hapr
oxy.plugin_driver.HaproxyOnHostPluginDriver:default
```

Una vez configurado instalaremos el paquete necesario para la utilización del plugins de LbasS:

```
apt-get install neutron-lbaas-agent
```

Configuraremos lbass en el fichero /etc/neutron/plugins/lbaas_agent.ini, en el fichero le definiremos sus drivers y el haproxy que es lo que utilizaremos para balancear la carga:

```
[DEFAULT]
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
device_driver =
neutron.services.loadbalancer.drivers.haproxy.namespace_driver.HaproxyNSDriver

[haproxy]
user_group = haproxy
```

Configuraremos fwass en el fichero /etc/neutron/plugins/fwaas_driver.ini, en este fichero habilitaremos el servicio y definiremos el driver de iptables que es el que vamos a utilizar para el firewall de openstack

```
[fwaas]
enabled = True
driver = neutron.services.firewall.drivers.linux.iptables_fwaas.IptablesFwaasDriver
```

Para finalizar la parte de neutrón reiniciaremos todos sus servicios para dejar fijados todos los servicios instalados:

```
service neutron-dhcp-agent restart
service neutron-l3-agent restart
service neutron-lbaas-agent restart
service neutron-lbaas-agent restart
service neutron-metadata-agent restart
service neutron-plugin-openvswitch-agent restart
```

3.3.5. Cinder.

OpenStack Block Storage (Cinder) proporciona dispositivos de almacenamiento a nivel de bloque persistentes para usar con instancias de OpenStack Compute.

El sistema de almacenamiento de bloques gestiona la creación, aplicación y el desprendimiento de los dispositivos de bloque a los servidores. Volúmenes de almacenamiento de bloque se integran plenamente en OpenStack Compute y el Dashboard que permite a los usuarios en la nube gestionar sus propias necesidades de almacenamiento. Además del almacenamiento del servidor local de Linux, puede utilizar las plataformas de almacenamiento incluyendo Ceph, CloudByte, Coraid, EMC (VMAX y VNX), GlusterFS, Hitachi Data Systems, IBM Storage (familia Storwize, controlador de volumen SAN, XIV Storage System, y GPFS) , Linux LIO, NetApp, Nexenta, Scality, SolidFire, HP (StoreVirtual y 3PAR StoreServ familias) y almacenamiento puro.

El almacenamiento de bloques es apropiado para escenarios donde el rendimiento es sensible, tales como el almacenamiento de base de datos, sistemas de archivos expandibles, o la prestación de un servidor con acceso al almacenamiento a nivel de bloque en bruto. La gestión Snapshot ofrece una potente funcionalidad para realizar copias de seguridad de los datos guardados en volúmenes de almacenamiento en bloque. Las instantáneas no se pueden restaurar ni utilizar para crear un nuevo volumen de almacenamiento en bloque.

La API de Cinder permite la manipulación de volúmenes, tipos de volúmenes y *snapshots*. cinder-api acepta los pedidos y los enruta a cinder-volume para ser procesados.

- cinder-volume reacciona leyendo o escribiendo a la base de datos que mantiene el estado, interactúa con otros procesos (como el cinder-scheduler) a través de la cola de mensajes y directamente actúa sobre el almacenamiento proveyendo hardware o software.
- cinder-scheduler selecciona el nodo para el almacenamiento por bloques óptimo para la creación de un volumen sobre él.
- La cola de mensajes encamina información entre los procesos de Cinder.
- Una base de datos almacena el estado de los volúmenes.

3.3.5.1. Instalación de cinder

Volviendo al inicio al punto de la creación del laboratorio, recordaremos que hemos creado la estructura necesaria para poder trabajar con volúmenes dandole almacenamiento a los dos servidores Virtualizador01 y Virtualizador02, como esto ya está realizado procederemos a instalar los paquetes necesarios en las siguientes máquinas.

- *En supervisor:*

apt-get install cinder-api cinder-scheduler

- *En virtualizador01 y virtualizador02*

apt-get install cinder-volume python-mysqldb

Una vez instalado los paquetes procederemos a configurar la base de datos para el servicio cinder:

Borraremos el fichero que genera por defecto para el sqlite

rm /var/lib/cinder/cinder.sqlite

Crearemos la base de datos en nuestro MariaDB:

CREATE DATABASE cinder;
GRANT ALL ON cinder.* TO 'cinderuser'@'localhost' IDENTIFIED BY 'cinderpass';

Daremos permisos sobre la base de datos a los dos virtualizadores, esto es necesario porque estos servidores escribirán en la base de datos:

GRANT ALL ON cinder.* TO 'cinderuser'@150.200.180.101 IDENTIFIED BY 'cinderpass';
GRANT ALL ON cinder.* TO 'cinderuser'@150.200.180.102 IDENTIFIED BY 'cinderpass';

Configuraremos el acceso de la base de datos en supervisor en el fichero /etc/cinder/cinder.conf

[database]
connection = mysql://cinderuser:cinderpass@localhost/cinder

Configuraremos el acceso a la base de datos (que reside en la máquina supervisor) en los ficheros de configuración /etc/cinder/cinder.conf de las máquinas virtualizador01 y virtualizador02:

```
[database]
connection = mysql://cinderuser:cinderpass@150.200.180.100/cinder
```

Una vez configurado precederemos a reiniciar los servicios de la api y del scheduler de la máquina supervisor:

```
service cinder-api restart
service cinder-scheduler restart
```

Y ya cogidos los cambios generaremos la base de datos:

```
cinder-manage db sync
```

3.3.5.2.Creación de servicio y usuario.

Una vez creada la base de datos procederemos a crear el servicio cinder en la máquina ostra.it.uc3m.es

keystone service-create --name=cinder --type=volume --description="Block Storage Service"

De esta ejecución sacaremos el valor CINDER_SERVICE_ID porque lo necesitaremos para la definirle porque url nos va a gestionar keystone las órdenes cuando le definimos su endpoint:

keystone endpoint-create --region Madrid --service-id=<CINDER_SERVICE_ID> --publicurl=http://150.200.190.100:8776/v1/%\(tenant_id\)s --internalurl=http://150.200.180.100:8776/v1/%\(tenant_id\)s --adminurl=http://150.200.180.100:8776/v1/%\(tenant_id\)s

Una vez creado el servicio crearemos el usuario en keystone:

keystone user-create --name cinder --pass c1nd3r --email tio_xexu@yahoo.es

Comprobaremos que está bien creado

```
root@ostra:~# keystone user-list
+----------------------------------+---------+---------+--------------------+
|                id                |   name  | enabled |       email        |
+----------------------------------+---------+---------+--------------------+
| 97f4fef867e44fd28333e1046c901002 |  admin  |   True  | tio_xexu@yahoo.es  |
| ff83ec4fdf1942528a3a3c349fbc12e5 |  cinder |   True  | tio_xexu@yahoo.es  |
| 119e8e0d44594bfbb5a4958de35ff9c3 |  glance |   True  | tio_xexu@yahoo.es  |
| 8c72d9f81b534329aeeb371523b2d1c6 | neutron |   True  | tio_xexu@yahoo.es  |
| 47ed1c8fb05c48039ab71313fe0a6097 |   nova  |   True  | tio_xexu@yahoo.es  |
+----------------------------------+---------+---------+--------------------+
```

Crearemos el rol:

keystone user-role-add --tenant service --user cinder --role admin

3.3.5.3.Configuración Cinder.

Una vez creado el servicio cinder procederemos a configurarlo en las tres máquinas que intervienen en el servicio:

1º *Supervisor*.

Hay que configurar en el fichero /etc/cinder/cinder.conf

- La autenticación del servicio de cinder a través de keystone (le diremos usuario, password y url por donde escucha):

```
auth_strategy = keystone

[keystone_authtoken]
auth_uri = http://150.200.180.100:5000/v2.0
identity_uri = http://150.200.180.100:35357
admin_tenant_name = service
admin_user = cinder
admin_password = c1nd3r
```

- Configuración del servicio cinder, en este punto le definiremos sus ficheros de configuración, por donde escucha la peticiones (IP y puerto) y en que path queremos configurarlo.

```
verbose = True
rootwrap_config = /etc/cinder/rootwrap.conf
api_paste_confg = /etc/cinder/api-paste.ini

osapi_volume_listen = 0.0.0.0
osapi_volume_listen_port = 8776

volume_name_template = volume-%s

state_path = /var/lib/cinder
lock_path = /var/lock/cinder
```

- Configuración de RabbitMq

```
rpc_backend = rabbit
rabbit_host = 150.200.180.100
rabbit_port = 5672
rabbit_userid = openstackuser
rabbit_password = openstackpass
rabbit_virtual_host = /openstack
```

- Procedemos a reiniciar los servicios para que coja el cambio:

```
service cinder-api restart
service cinder-scheduler restart
```

2º *Virtualizador01 y Virtualizador02*

Hay que configurar el fichero /etc/cinder/cinder.conf

- Configuración del servicio cinder:

```
verbose = True
rootwrap_config = /etc/cinder/rootwrap.conf
api_paste_confg = /etc/cinder/api-paste.ini

iscsi_helper = tgtadm
volume_group = cinder-volumes
volumes_dir = /var/lib/cinder/volumes

state_path = /var/lib/cinder
lock_path = /var/lock/cinder
```

- Tendremos que definir que gestor de colas utilizamos (en nuestro caso rabbit-mq), donde se encuentra (ip y puerto) y con que usuario en el fichero de configuración

```
rpc_backend = rabbit
rabbit_host = 150.200.180.100
rabbit_port = 5672
rabbit_userid = openstackuser
rabbit_password = openstackpass
rabbit_virtual_host = /openstack
```

- Procedemos a reiniciar los servicios para que coja el cambio:

```
service cinder-volume restart
```

3.3.5.4.Configuración de servicios extras.

Una vez ya configurado vamos a crear dos extras que proporciona cinder. Uno son las zonas y otro son los tipos de almacenamiento.

3.3.5.4.1 Zonas.

Una zona de disponibilidad en openstack es una partición lógica de máquinas o servicios de volumen dentro de un único despliegue de openstack. Las zonas de disponibilidad se definen a nivel de configuración de cada máquina, lo que proporciona un método para segmentar los nodos de cálculo por criterios arbitrarios, como características de hardware, ubicación física o designación de tarea.

Vamos a crear 2 zonas una en cada virtualizador, para poder diferenciarlas:

-Virtualizador01.

- Añadimos esta línea en el fichero /etc/cinder/cinder.conf

storage_availability_zone = zone01

- Reiniciamos el servicio:

service cinder-volume restart

-Virtualizador02.

- Añadimos esta línea en el fichero /etc/cinder/cinder.conf

storage_availability_zone = zone02

- Reiniciamos el servicio:

service cinder-volume restart

Ahora en la máquina supervisor tendremos que definir una zona por defecto, para que sin tener que definirlo se cree en esta zona:

- Añadimos esta línea en el fichero /etc/cinder/cinder.conf

default_availability_zone = zone01

- Reiniciamos los servicios:

service cinder-api restart
service cinder-scheduler restart

Para ver que se han creado bien listamos las zonas:

```
root@ostra:~# cinder availability-zone-list
+--------+-----------+
|  Name  |   Status  |
+--------+-----------+
| zone01 | available |
| zone02 | available |
+--------+-----------+
```

3.3.5.4.2 Tipos

El tipo de almacenamiento se utiliza sobre todo para diferenciar almacenamiento más barato y más barato, esto se suele definir por el tipo de redundancia del almacenamiento y por el tiempo de respuesta en lectura de una solicitud de un dato.

Aunque nosotros no tenemos dos tipos de discos distintos vamos a hacer un ejemplo y para poder diferenciar tipos tenemos que tener 2 almacenamientos, por lo que tendremos que darle otro almacenamiento a virtualizador01 y virtualizador02

```
virsh vol-create-as --pool vms --name virtualizador01-data02.img --capacity 25G --format
qcow2
virsh attach-disk virtualizador01 /repositorio/virtualizador01-data02.img --target vdc --
driver qemu --subdriver qcow2 --config --live

virsh vol-create-as --pool vms --name virtualizador02-data02.img --capacity 25G --format
qcow2
virsh attach-disk virtualizador02 /repositorio/virtualizador02-data02.img --target vdc --
driver qemu --subdriver qcow2 --config --live
```

En ambos crearemos ahora el volumen:

```
pvcreate /dev/vdc
vgcreate cinder-ssd /dev/vdc
```

Ya dado el disco nuevo a ambas máquinas vamos a configurar los tipos en el fichero /etc/cinder/cinder.conf de virtualizador01 y virtualizador02:

Primero le diremos que utiliza distintos tipos de discos y los definimos por el nombre que le vayamos a dar, para después definirlos en cinder y relacionar que disco es cada uno:

```
[DEFAULT]
...
#volume_group = cinder-volumes

enabled_backends = lvm-sata,lvm-ssd

[lvm-sata]
volume_group = cinder-volumes
volume_backend_name = sata


[lvm-ssd]
volume_group = cinder-ssd
volume_backend_name = ssd
```

Una vez hecho los cambios en los virtualizadores procederemos a reiniciar el servicio para que coja los cambios:

service cinder-volume restart

En la máquina supervisor definiremos en el fichero /etc/cinder/cinder.conf cuál de los dos tipos es el por defecto.

default_volume_type = silver

Una vez realizado el cambio reiniciaremos los servicios en supervisor:

service cinder-api restart
service cinder-scheduler restart

Ahora crearemos los tipos desde la máquina ostra.it.uc3m.es

cinder type-create gold
cinder type-key gold set volume_backend_name=ssd

cinder type-create silver
cinder type-key silver set volume_backend_name=sata

3.3.5.4.3 ISCSI.

iSCSI (Abreviatura de *Internet SCSI*) es un estándar que permite el uso del protocolo SCSI sobre redes TCP/IP. iSCSI es un protocolo de la capa de transporte definido en las especificaciones SCSI-3. Otros protocolos en la capa de transporte son SCSI Parallel Interface y canal de fibra.

La adopción del iSCSI en entornos de producción corporativos se ha acelerado en estos momentos gracias al aumento del Gigabit Ethernet. La fabricación de almacenamientos basados en iSCSI (red de área de almacenamiento) es menos costosa y está resultando una alternativa a las soluciones SAN basadas en Canal de fibra.

Como vimos en el dibujo del laboratorio dejamos una red para datos, esta red en teoría seria para poder configurar almacenamiento ISCSI a las máquinas virtuales. Configuremos un ejemplo porque para el aprendizaje creemos que es un ejemplo acertado por su gran uso, aunque no tengamos almacenamiento por iSCSI para poder hacer las pruebas con el directamente.

Configuraremos que funcione el iscsi en Virtualizador01 y Virtualizador02 en /etc/cinder/cinder.conf.

iscsi_ip_address = 10.222.0.101

Y en /etc/nova/nova.conf, le diremos que vamos a utilizar cinder con iscsi.

cinder_catalog_info = volume:cinder:internalURL

Una vez cambiado reiniciamos los servicios:

service cinder-volume restart
service nova-compute restart

3.3.5.4.4 Glance con cinder

Por último configuraremos el glance con cinder.

En virtualizador01 y virtualizador02 añadiremos en el fichero /etc/cinder/cinder.conf.

auth_strategy = keystone

glance_host = 150.200.180.100
glance_port = 9292

Una vez cambiado reiniciaremos los servicios:

service cinder-volume restart

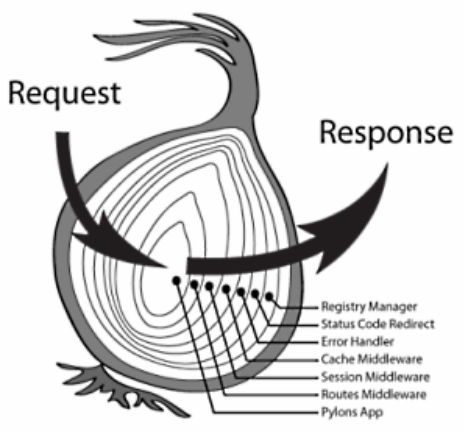
Una vez configurados comprobaremos los servicios desde ostra.it.uc3m.es:

```
root@ostra:~# cinder service-list
+------------------+--------------------------+--------+---------+-------+----------------------------+-----------------+
|      Binary      |           Host           |  Zone  |  Status | State |         Updated_at         | Disabled Reason |
+------------------+--------------------------+--------+---------+-------+----------------------------+-----------------+
| cinder-scheduler |        supervisor        |  nova  | enabled |   up  | 2015-07-01T16:41:23.000000 |       None      |
|  cinder-volume   |     virtualizador01      | zone01 | enabled |  down | 2015-04-04T08:12:12.000000 |       None      |
|  cinder-volume   | virtualizador01@lvm-sata | zone01 | enabled |   up  | 2015-07-01T16:41:23.000000 |       None      |
|  cinder-volume   | virtualizador01@lvm-ssd  | zone01 | enabled |   up  | 2015-07-01T16:41:14.000000 |       None      |
|  cinder-volume   |     virtualizador02      | zone02 | enabled |  down | 2015-04-04T08:12:22.000000 |       None      |
|  cinder-volume   | virtualizador02@lvm-sata | zone02 | enabled |   up  | 2015-07-01T16:41:20.000000 |       None      |
|  cinder-volume   | virtualizador02@lvm-ssd  | zone02 | enabled |   up  | 2015-07-01T16:41:20.000000 |       None      |
+------------------+--------------------------+--------+---------+-------+----------------------------+-----------------+
```

3.3.6. Horizon.

El Dashboard de OpenStack (Horizont) proporciona a los administradores y usuarios una interfaz gráfica para el acceso, la provisión y automatización de los recursos basados en la nube. El diseño permite que los productos y servicios de terceros, tales como la facturación, el monitoreo y las herramientas de gestión adicionales. El Dashboard es sólo una forma de interactuar con los recursos de OpenStack. Los desarrolladores pueden automatizar el acceso o construir herramientas para gestionar sus recursos mediante la API nativa de OpenStack o la API de compatibilidad EC2.

- Es una aplicación web Django, un web framework para perfeccionistas ;) Todos aquellos que estén familiarizados con este framework encontrarán el código de Horizon muy fácil de comprender.
- Se encuentra implementado a través de mod_wsgi en Apache.
- mod_wsgi implementa un módulo de Apache simple de usar capaz de alojar cualquier aplicación desarrollada en Python que soporte la interfaz WSGI.
- *Web Server Gateway Interface* WSGI es una especificación para que servidores de aplicaciones y servidores web se comuniquen con aplicaciones web.



WSGI middleware – por Pylons

- El código se encuentra divido en módulos reutilizables con la lógica, interacción con las APIs, y la presentación, para facilitar la personalización en diferentes sitios.
- También posee una pequeña base de datos, por defecto SQLite3, para algunas opciones, pero la mayoría de los datos son provistos por los otros servicios.
- Horizon es una implementación del Dashboard, no el Dashboard en sí mismo. Pueden hacerse distintas implementaciones que se ajusten a las necesidades del usuario.

3.3.6.1. Instalación Horizon

El primer paso es la instalación de paquetes para Horizon en supervisor.

apt-get install openstack-dashboard apache2 libapache2-mod-wsgi memcached python-memcache

Luego procedemos a desinstalar el template de Ubuntu

apt-get remove --purge openstack-dashboard-ubuntu-theme

Procederemos configurar Keystone en /etc/openstack-dashboard/local_settings.py

OPENSTACK_HOST = "150.200.180.100"

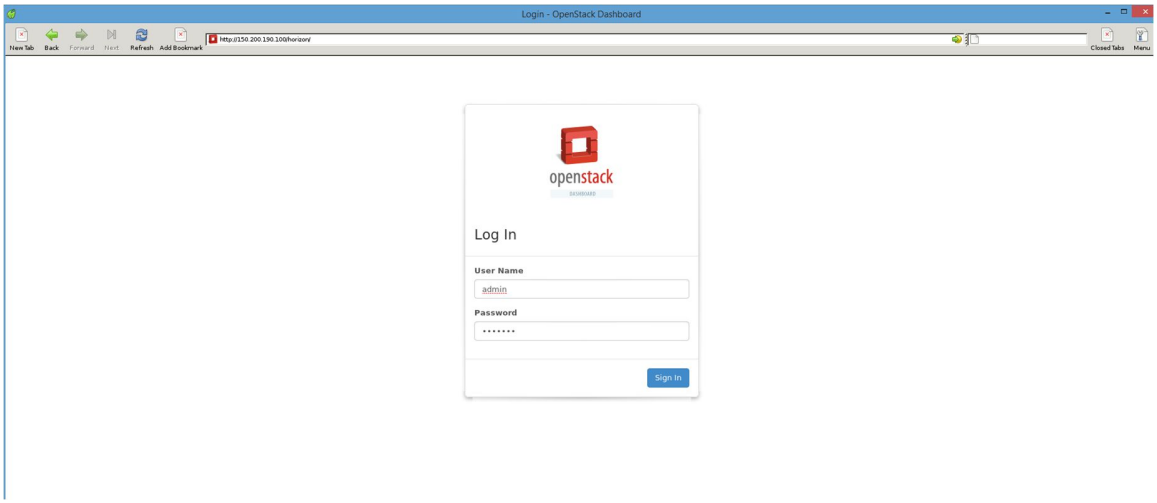
Reiniciamos el servidor web

service apache2 restart

Una vez instalado el acceso al panel web es:

http://150.200.190.100/horizon

user: admin
pass: s3cr3t0



4.- VALIDACIÓN

Una vez instalado ya todos los componentes de openstack solo nos falta validar su funcionamiento. En nuestro caso su validación se realizará a través de un pequeño laboratorio que simule un caso real de producción de una empresa. Para ello utilizaremos la herramienta instalada en el último lugar que es horizon, esta herramienta nos permitirá gestionar nuestro laboratorio e instalarlo sin problemas.

4.1. Pruebas.

Como hemos dicho anteriormente procederemos a instalar un entorno de pruebas, en nuestro caso hemos elegido un entorno empresarial clásico con 4 máquinas y 3 redes.

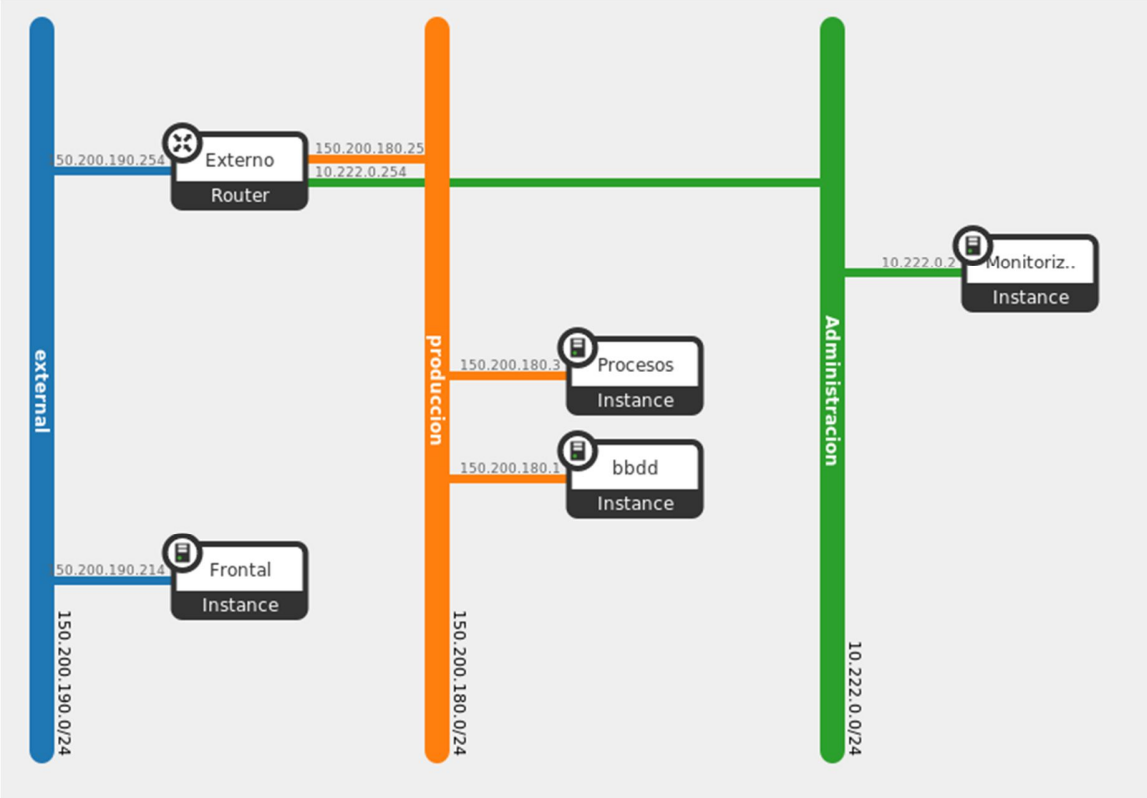
Las máquinas serán:

- Máquina frontal web, donde la empresa daría el acceso a su web desde el exterior.
- Máquina bbdd donde la web accedería a consultar en la bbdd los datos de la empresa.
- Máquina de proceso, esta estará encargada de generar y cargar datos en la base de datos en background, a través de distintos programas o procesos internos.
- Máquina de monitorización, estará encargada de monitorizar el buen funcionamiento de las otras 3 máquinas de la empresa.

Las redes serán:

- Red External, esta red es donde se conecta al exterior la empresa.
- Red Producción, red segura a la que no se puede acceder desde el exterior de la empresa.
- Red Administración, red donde se encuentra el monitor.

Adjunto el esquema que vamos a lograr:



4.1.1. Redes:

Empezaremos creando las 3 redes que vamos a tener. La red External, ya aparecerá creada, si recordamos los pasos anteriores para realizar las pruebas de funcionamiento de neutrón ya la creamos. Vuelvo a poner los comandos ejecutados anteriormente.

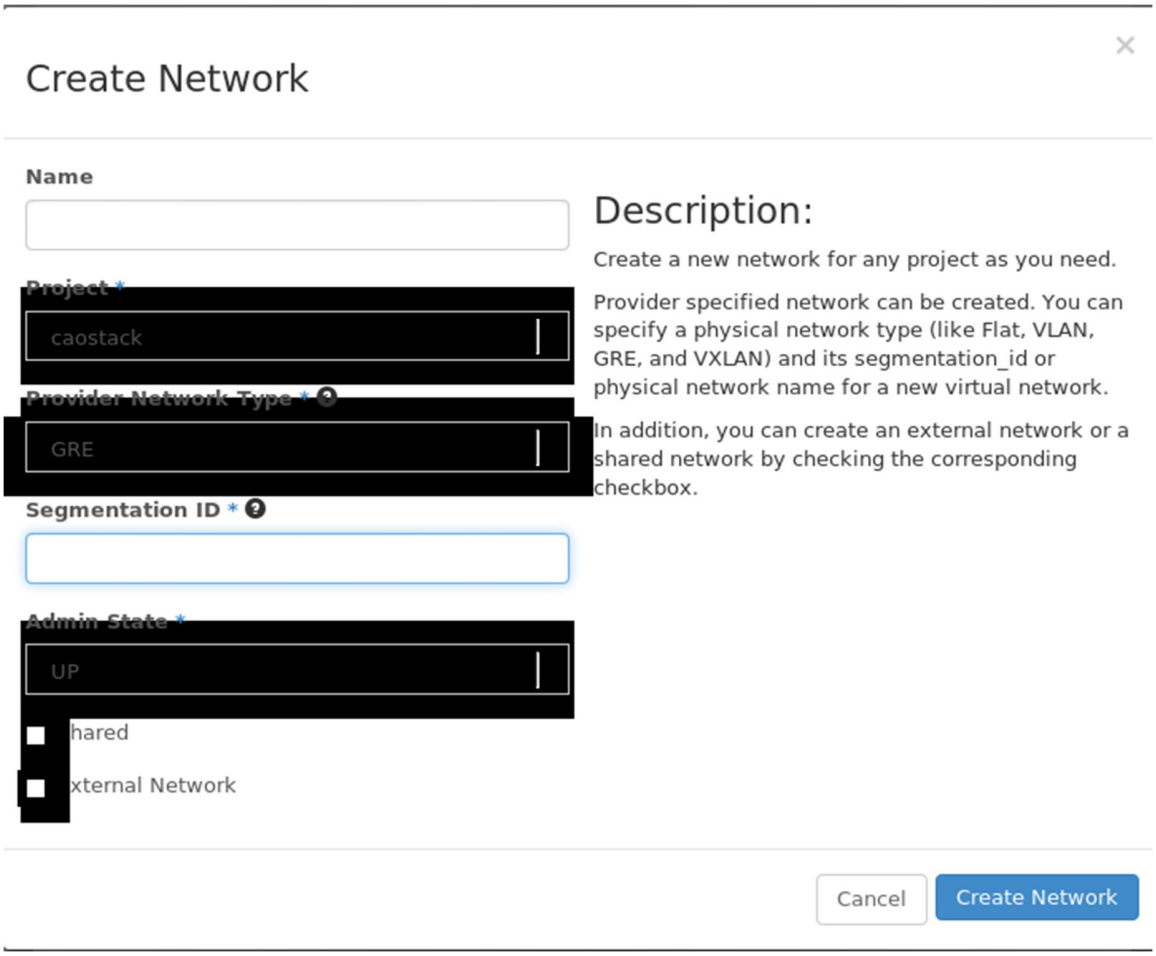
neutron net-create external --shared --router:external=True

Y después su subred:

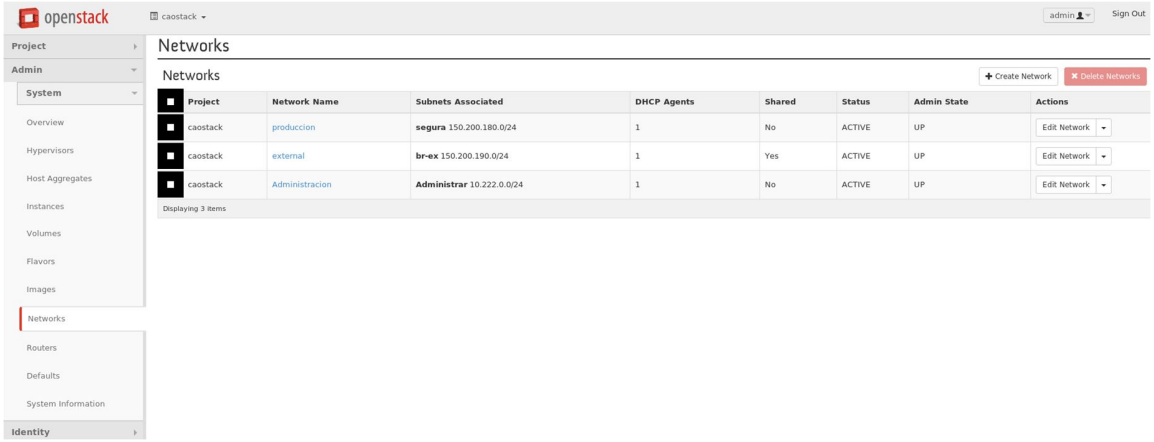
neutron subnet-create external --allocation-pool start=150.200.190.200,end=150.200.190.215 --gateway=150.200.190.254 --disable-dhcp 150.200.190.0/24

Como observareis primero se crea la red y luego se define su subred, en nuestro caso ya lo crearemos de modo gráfico con horizon.

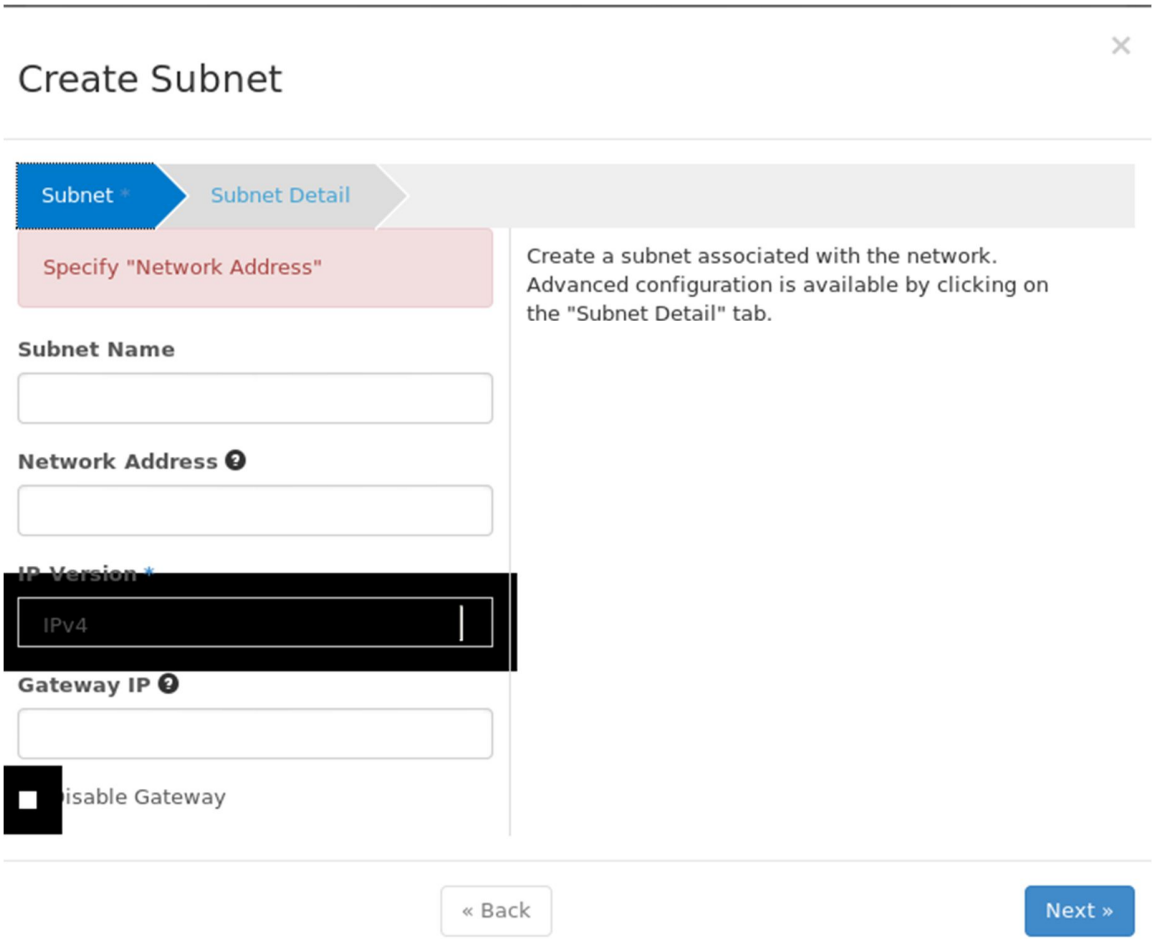
Ahora mostrare la ventana en horizon y los datos para crear una red nueva.



Con Horizon crearemos el resto de las redes que vamos a utilizar en el laboratorio. Una vez creadas veremos cómo Horizon muestra las 3 redes creadas y como quedan configuradas:

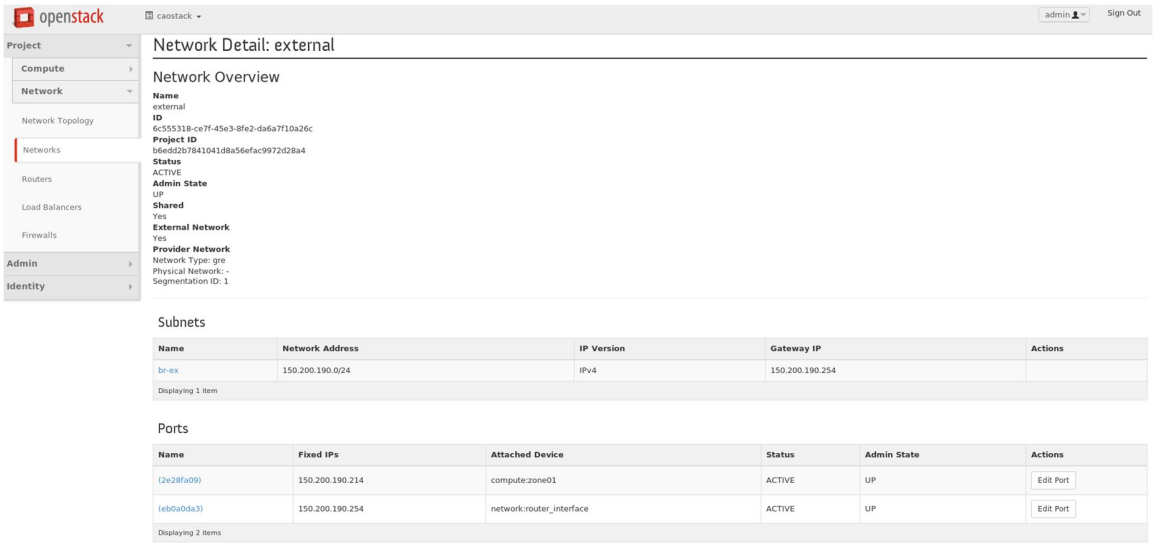


Una vez creada cada red deberemos crear la subred que tiene dentro la red para definir sus valores, sino no haría nada.

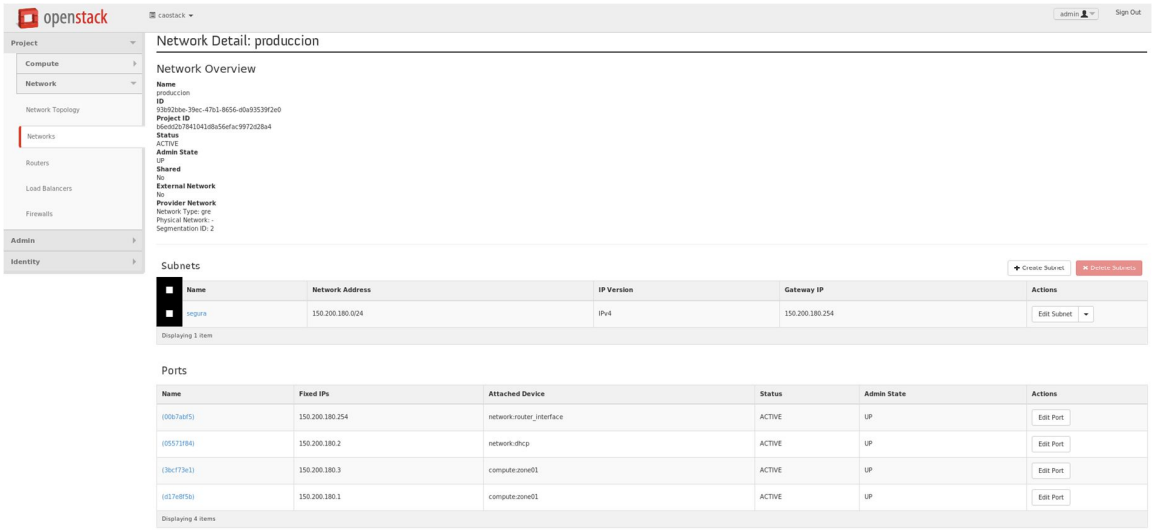


Ahora mostraremos como quedarían nuestras 3 redes a través del entorno gráfico, las mostraremos una a una:

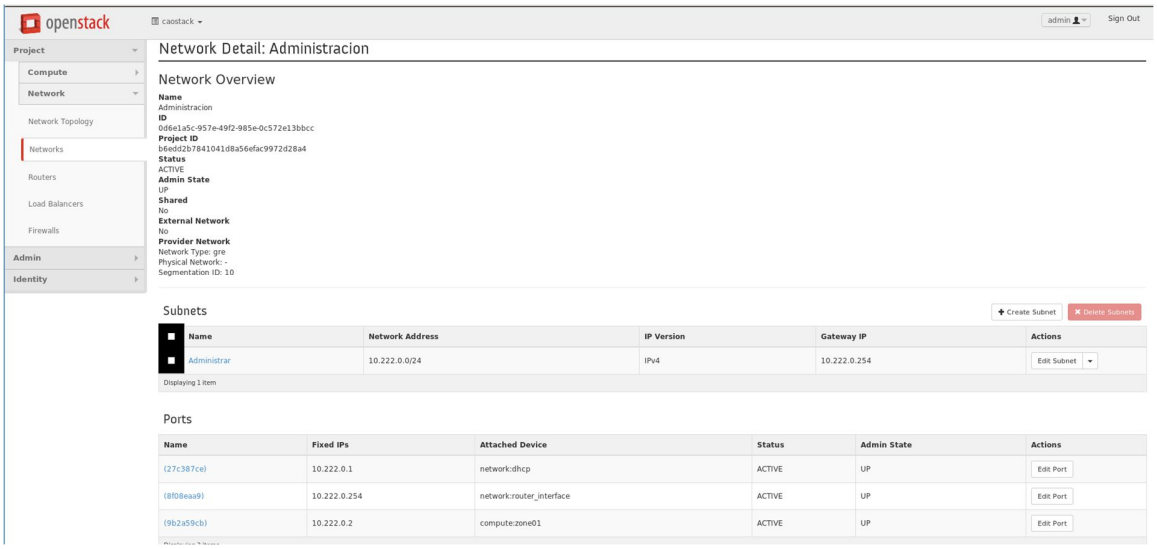
- External:



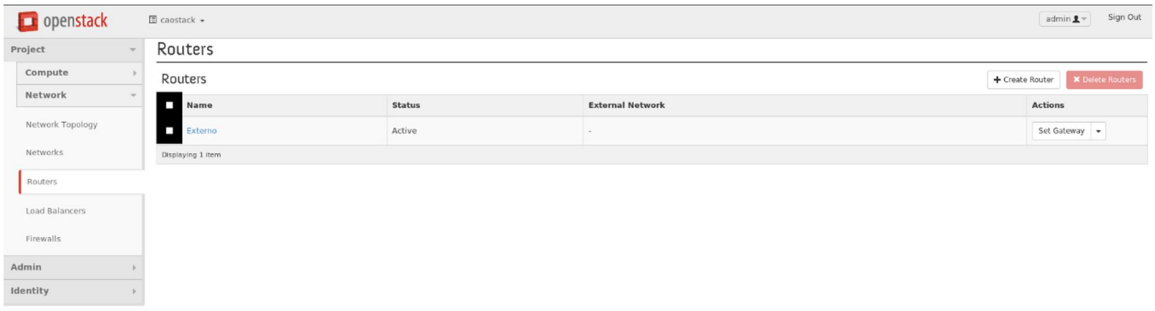
- Producción:



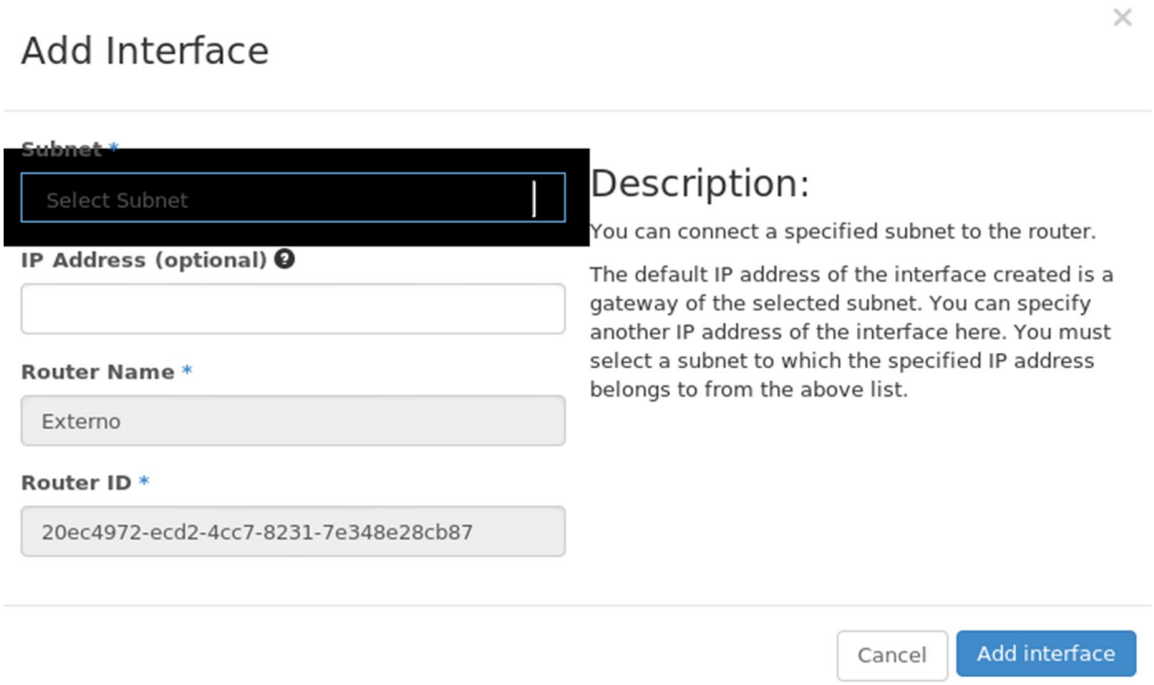
- Administración:



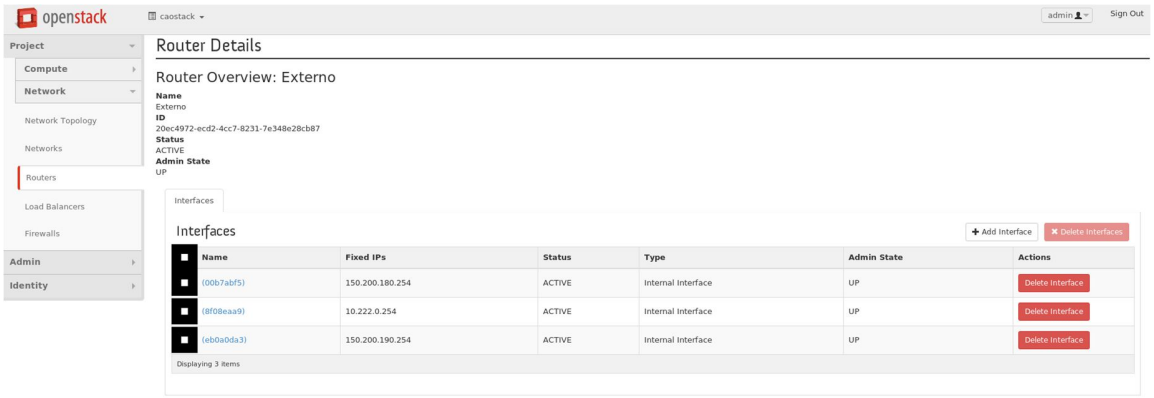
Una vez creada las redes necesitaremos crear un router para que puedan verse las redes entre sí. Por tanto el siguiente paso es crear un router, el paso de crearlo es muy sencillo ya que como en el caso de la creación de red se define y luego se configurará, se llamara externo y quedara así:



Luego toca el paso más importante que es configurarlo y crear sus gateways a todas las redes, como todas las redes que hemos creado son con una máscara de red de /24, tendremos que decidir cuál es su gateway, para unificar en nuestro caso hemos decidido que sea la ip 254 de la subred:

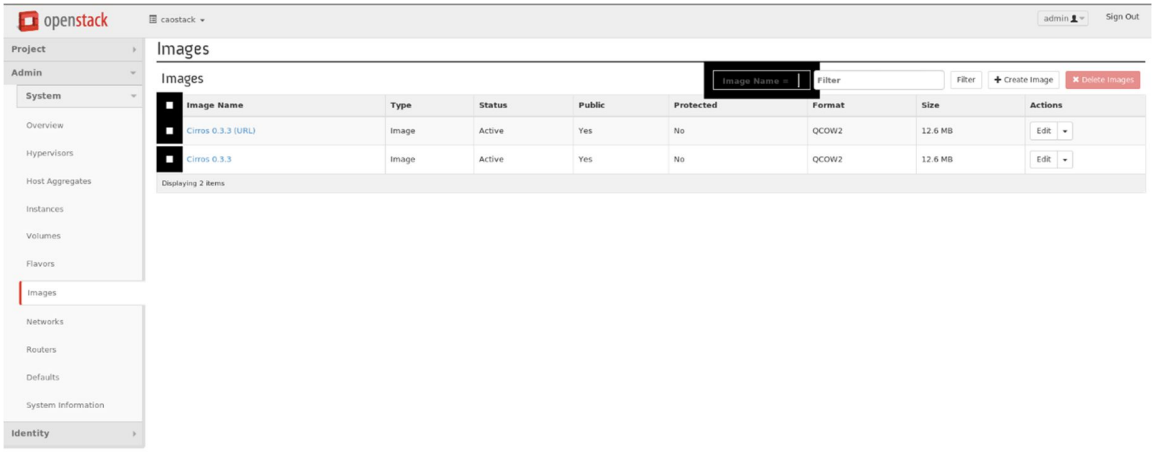


Después de configurar nuestro router y definir sus gateways para que se puedan comunicar las redes entre si este quedara así:



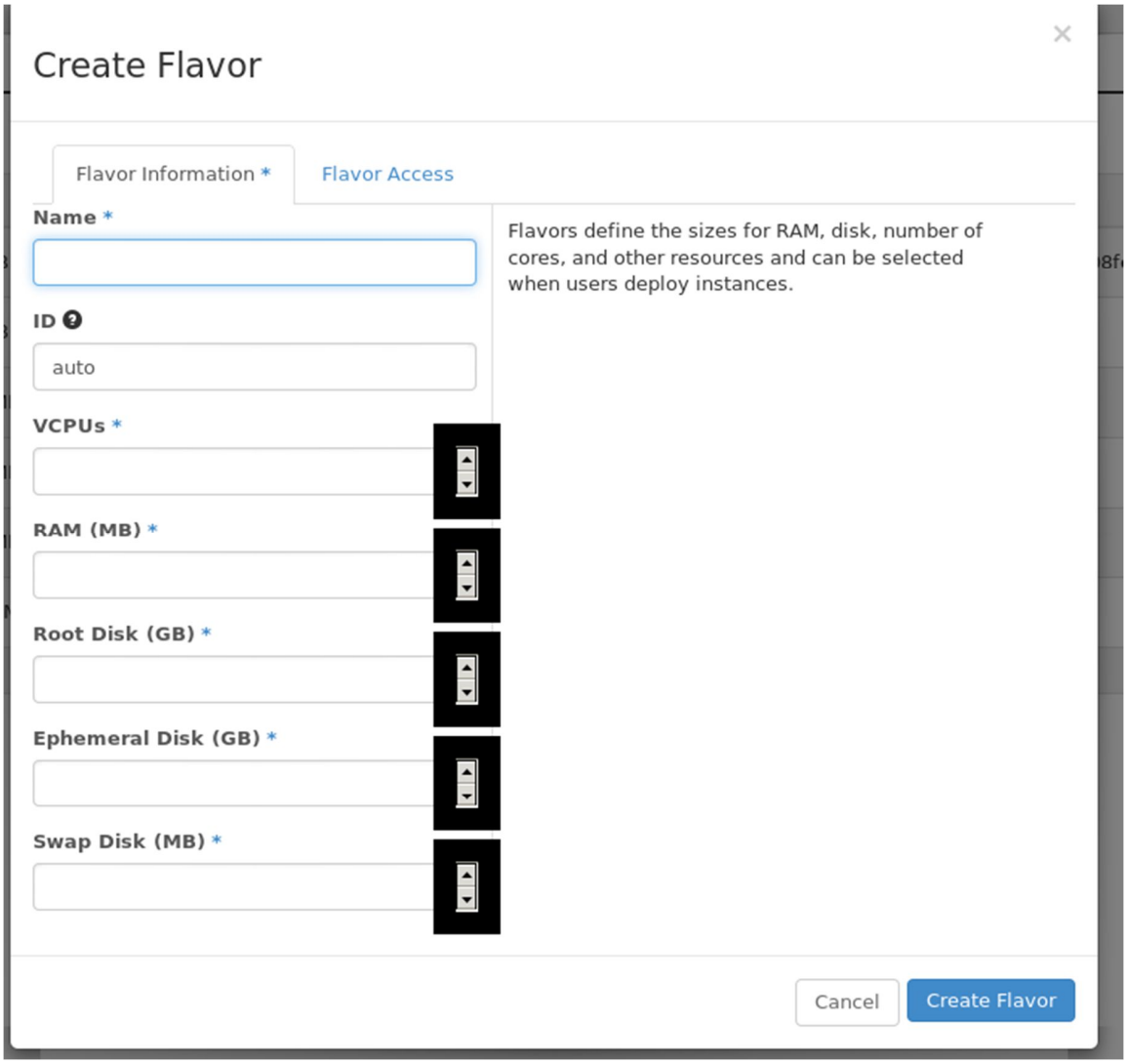
4.1.2. Máquinas:

Para crear las máquinas vamos a utilizar dos cosas que nos harán el trabajo más sencillo. El primero es la utilización de imágenes de sistemas operativos, en el punto de glance creamos un repositorio con dos imagines del sistema operativo cirros, utilizaremos estas imágenes para la instalación de las máquinas, así aparecerá en el entorno gráfico de Horizon:

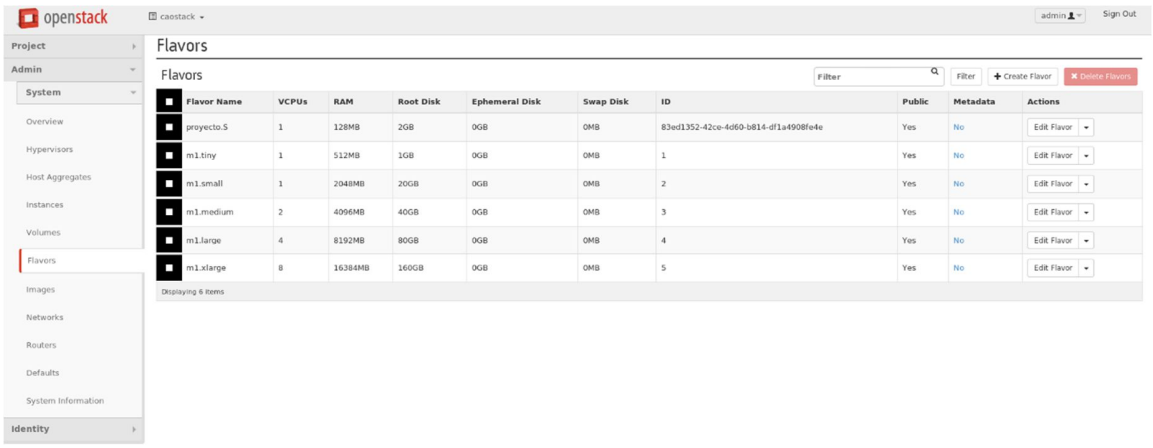


Además hay un concepto en openstack que se llama flavor (sabor) que te deja definir las características de las máquinas a crear, así puedes definir distintos tipos de máquinas dependiendo tus necesidades. Por defecto vienen unas estándar que te pueden valer para tu día a día, pero puedes crear unas plantillas de características especiales para ti. Nosotros hemos creado un flavor con unas características distintas para que los alumnos lo vean y aprendan cosas nuevas de openstack, para nuestro proyecto hemos creado un flavor que se llama proyecto.

Aquí mostramos como se crea un flavor de forma gráfica con Horizon:



Esto es como quedan los flavors de nuestro proyecto con los estándares y con nuestro nuevo flavor:



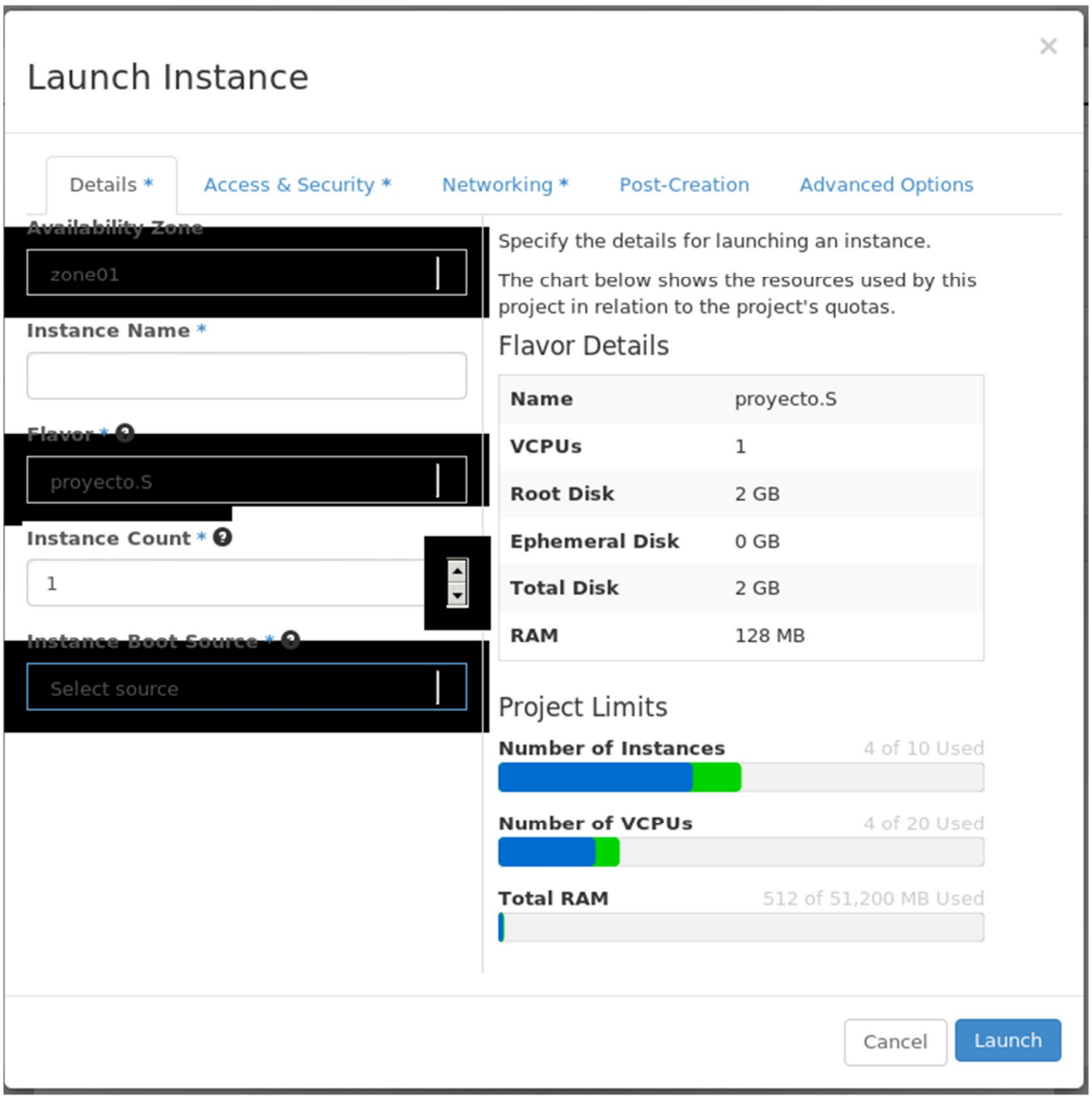
Con estas dos opciones empezaremos a crear nuestras máquinas de nuestro laboratorio.

Como tenemos servidores dhcp en cada subred (los creamos al crear la subred), al crear una máquina nueva y definirle las redes que va a utilizar, se instalará directamente con una IP asociada en cada IP.

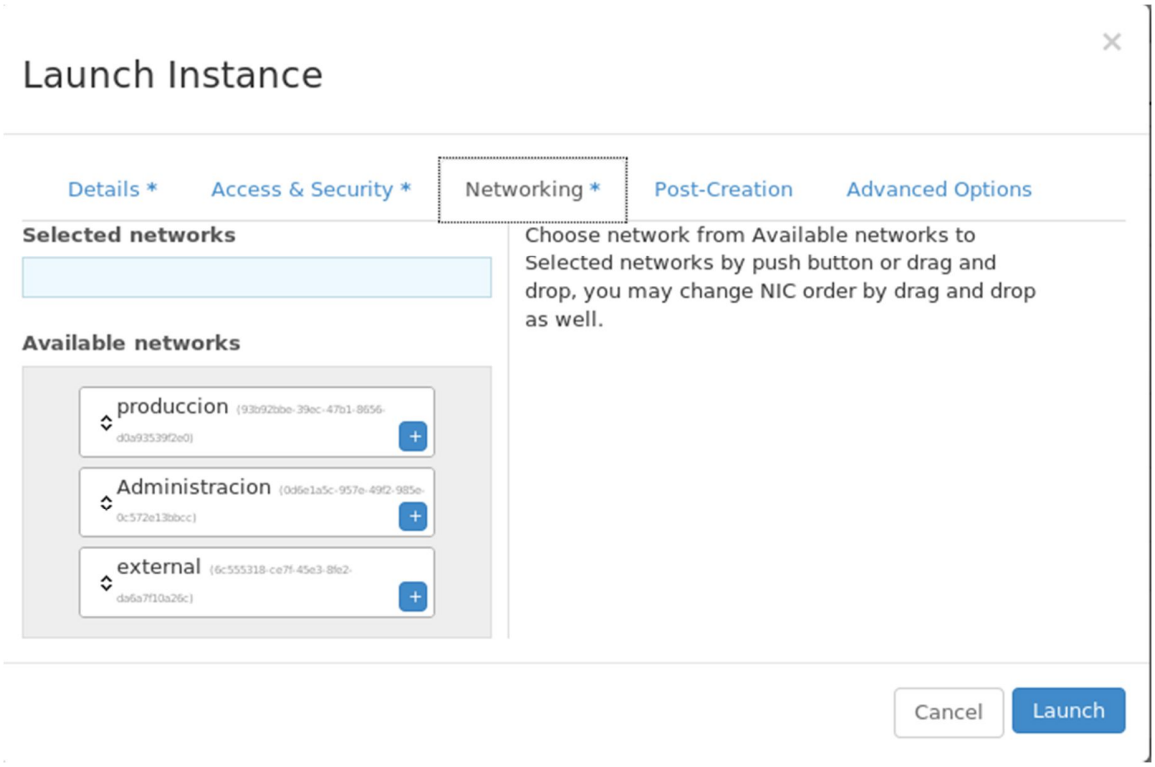
Al crear la máquina diciéndola que se instale desde una imagen de sistema operativo (en nuestro caso cirros), que tenemos cargada en glance el sistema arrancará con ella instalada.

El único caso distinto será la máquina Frontal, ya que debido a ser red externa tiene que estar caído el dhcp, nos asignara la IP pero luego tendremos que entrar en la máquina e configurárselo directamente en el fichero de configuración como una IP fija.

Vamos a crear una instancia, mostraremos las 2 pestañas más importantes en la creación de la máquina y es las que nosotros hemos modificado para la creación de las máquinas virtuales que son la relacionada con el sistema operativo, donde le indicas el flavor que vas a utilizar, el nombre de la máquina y desde que dispositivo vas a instalarle el sistema operativo, en nuestro caso la imagen que tenemos en glance:

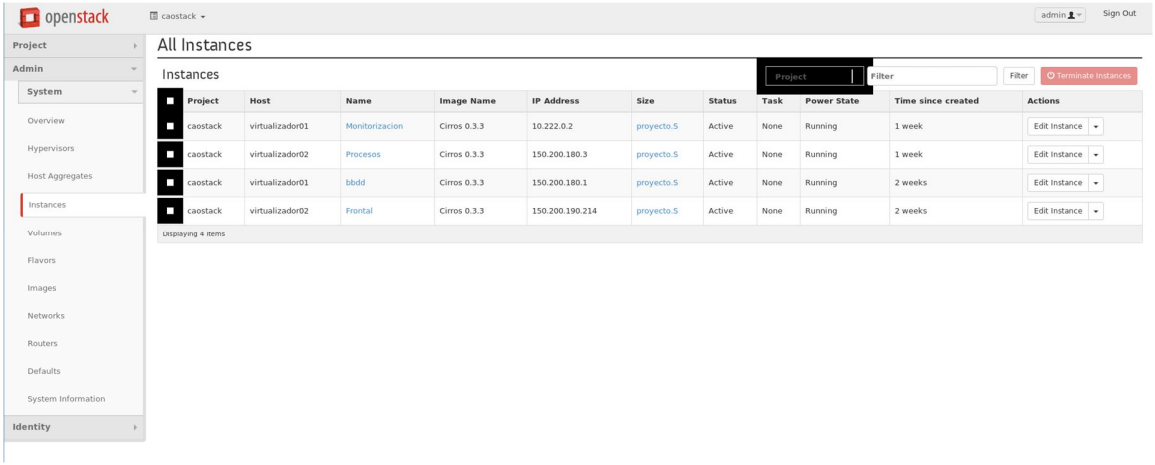


Y la de la configuración de la red, en esta configuración le diremos que redes va a tener la máquina a instalar, en nuestro caso te dará a mostrar las 3 redes que hemos creado en puntos anteriores que son producción, la red de administración y la red external como se ve en la siguiente captura de imagen:



Como se ve se puede configurar más puntos, pero estos son los dos puntos más importantes para hacer funcionar el laboratorio, el resto de los puntos complementarían de una forma más específica las máquinas instaladas, pero como esto es un laboratorio para la enseñanza no lo hemos visto necesario ya que son de menos uso.

Ahora mostramos como quedan las instancias que hemos creado para nuestro laboratorio:

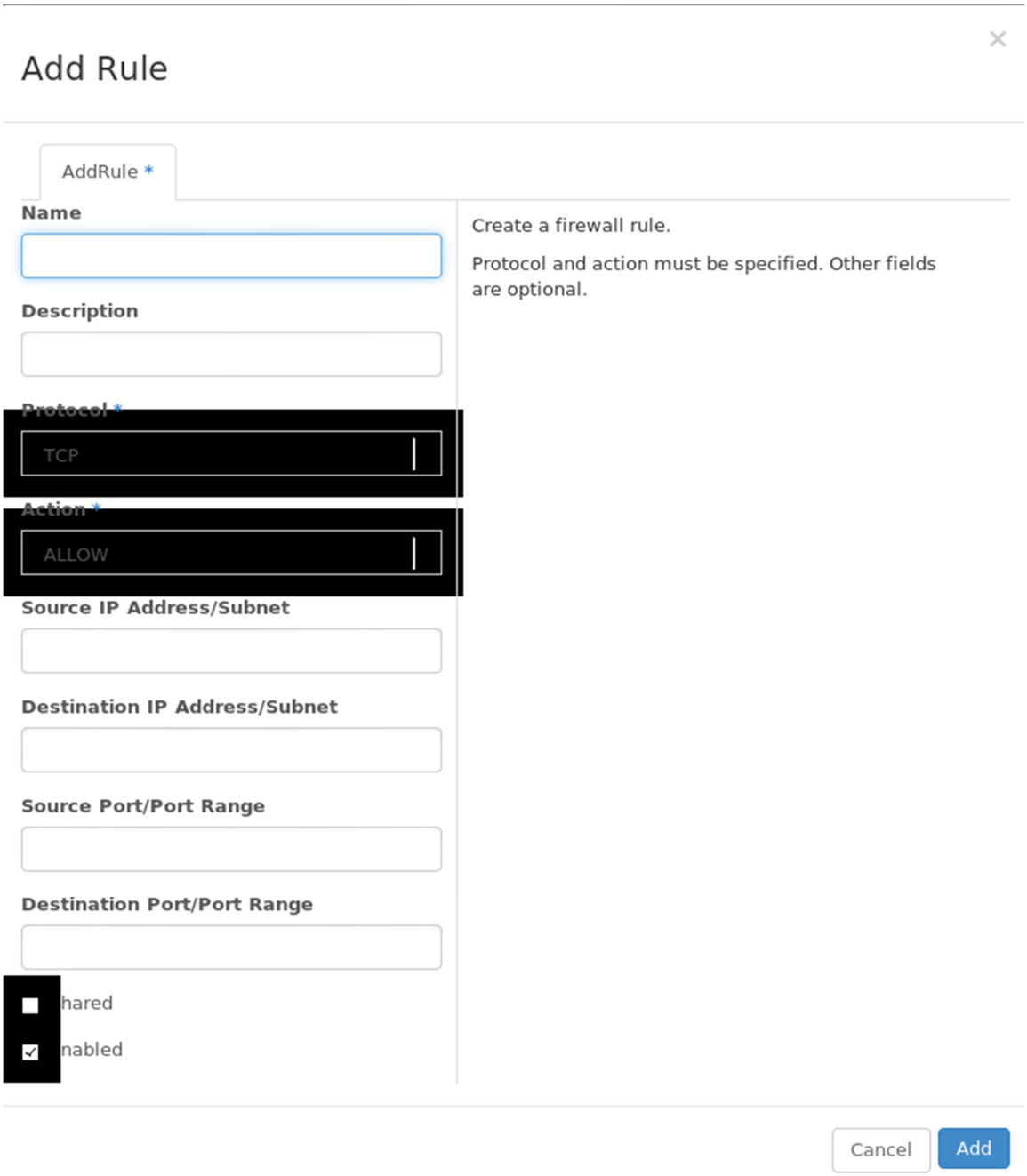


Ya creado el laboratorio solo nos quedaría la configuración de un firewall, como vimos en el punto de neutron instalamos en los extras el balanceador y el firewall, utilizaremos este último para configurar la red de la forma que dijimos.

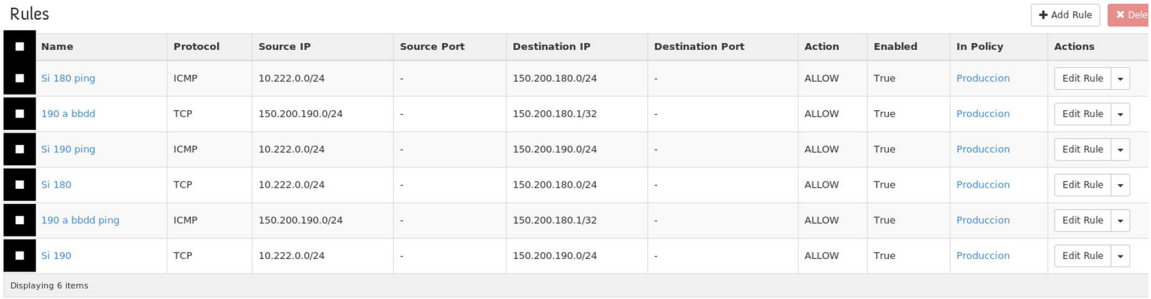
- La máquina externa solo ve a la máquina bbdd.
- La máquina bbdd ve a la máquina producción y viceversa.
- La máquina monitorización ve a todas y ninguna le ve.

Para esto tendremos que configurar las reglas del firewall, este firewall funciona con denegación a todo por defecto, así que habrá que crear reglas donde tengamos que llegar.

Para poder hacerlo funcionar, tendremos que crear las reglas, necesarias para lo que nosotros queremos en nuestro laboratorio:



Una vez creadas las reglas de nuestro laboratorio podremos verlas a través de Horizon y quedaran así:



Rules

+ Add Rule | ✖ Dele

Name	Protocol	Source IP	Source Port	Destination IP	Destination Port	Action	Enabled	In Policy	Actions
Si 180 ping	ICMP	10.222.0.0/24	-	150.200.180.0/24	-	ALLOW	True	Produccion	Edit Rule
190 a bbdd	TCP	150.200.190.0/24	-	150.200.180.1/32	-	ALLOW	True	Produccion	Edit Rule
Si 190 ping	ICMP	10.222.0.0/24	-	150.200.190.0/24	-	ALLOW	True	Produccion	Edit Rule
Si 180	TCP	10.222.0.0/24	-	150.200.180.0/24	-	ALLOW	True	Produccion	Edit Rule
190 a bbdd ping	ICMP	150.200.190.0/24	-	150.200.180.1/32	-	ALLOW	True	Produccion	Edit Rule
Si 190	TCP	10.222.0.0/24	-	150.200.190.0/24	-	ALLOW	True	Produccion	Edit Rule

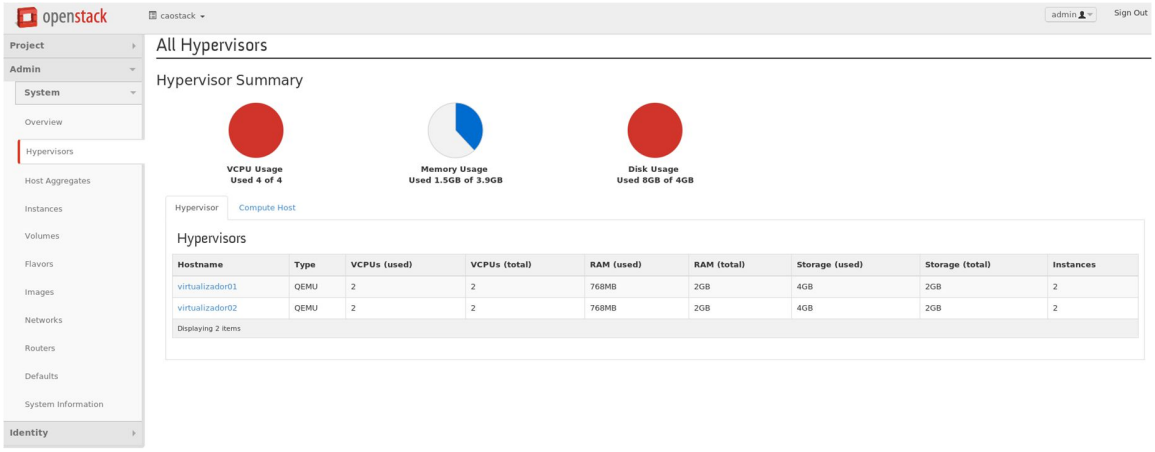
Displaying 6 items

Estas reglas las tendremos que meter en una política, se pueden definir varias políticas y tener algunas activas o inactivas, por esta razón hay que definirla, en nuestro caso al ser un ejemplo a nivel de enseñanza meteremos todas las reglas creadas en la misma política y esta política la activaremos en el firewall para que la red funcione como nosotros decidimos en el ejemplo creado, con estas reglas ya debería funcionar las máquinas como queremos.

4.2. Resultados.

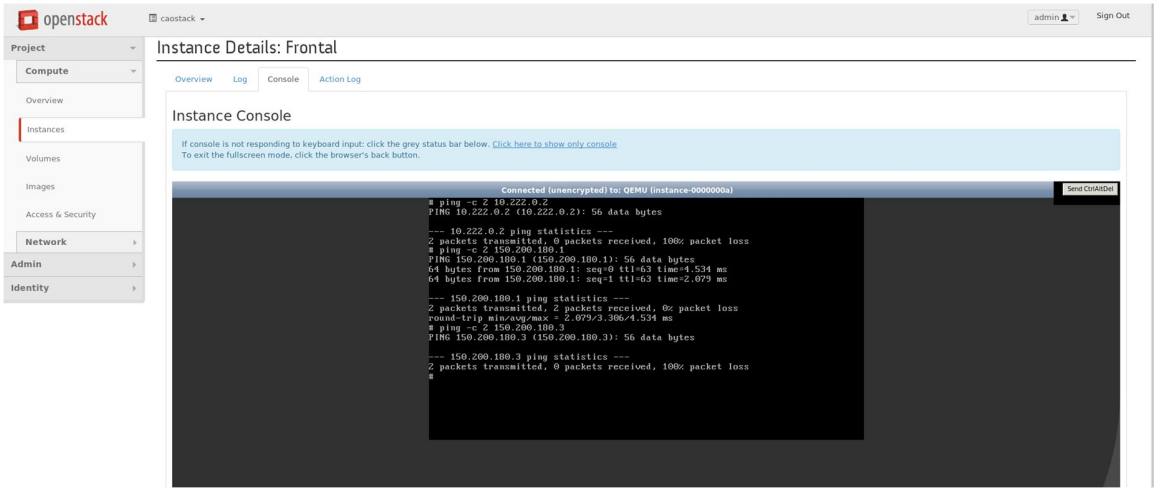
Una vez creado el laboratorio vamos a realizar una serie de comprobaciones para ver su funcionamiento.

A través de Horizon vamos a ver en que máquinas (hipervisores), en nuestro caso virtualizador01 y virtualizador02 están corriendo las instancias, como se ve en la gráfica están las máquinas balanceadas en los dos hipervisores habiendo 2 máquinas del laboratorio en cada hipervisor:

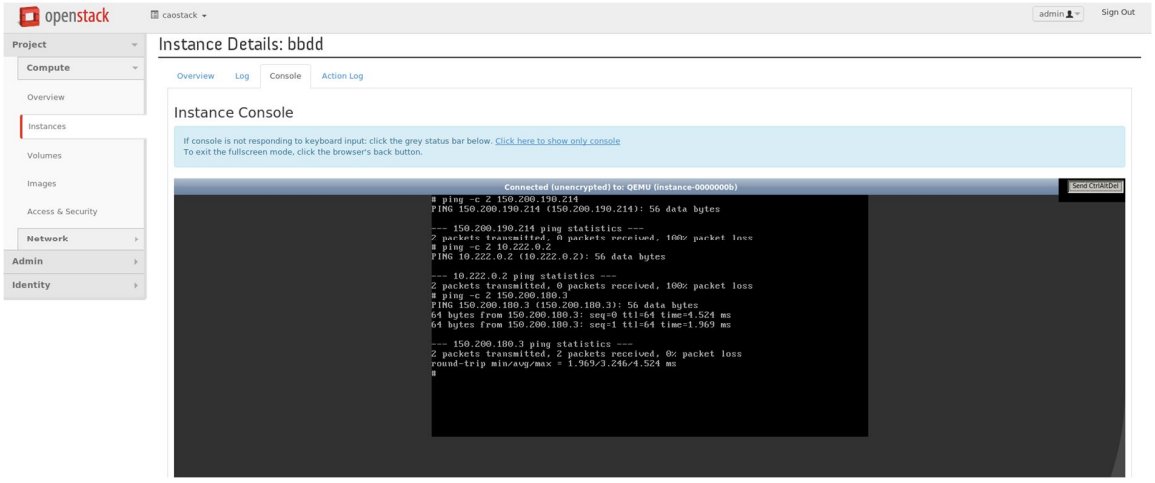


Una vez visto donde corren nuestras máquinas vamos a comprobar que se ven entre ellas como hemos configurado en el firewall y comprobaremos que las reglas introducidas funcionan realizando pings entre las máquinas.

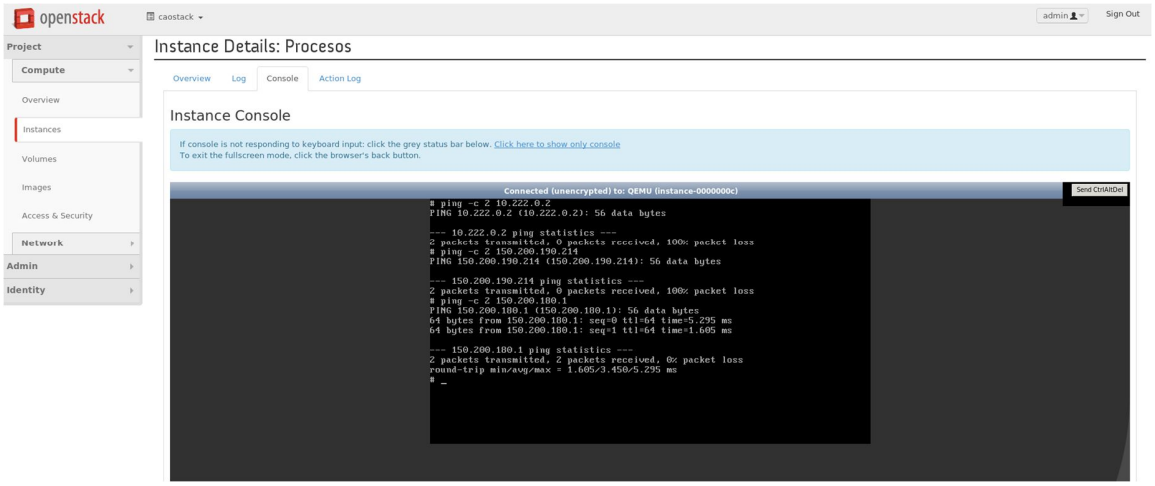
1º *Máquina Frontal.* Como se ve en el pantallazo siguiente cumple lo pedido y solo ve la máquina de bbdd, al resto de máquinas no puede acceder porque se lo impide el firewall del sistema.



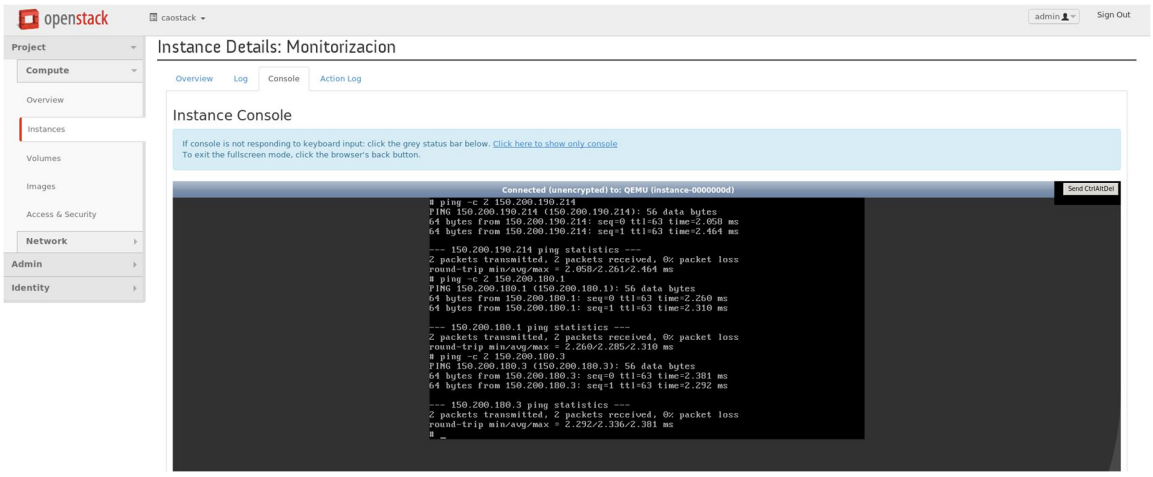
2º *Máquina bbdd.* Como se ve en el pantallazo siguiente cumple lo pedido, esta máquina solo ve a la máquina de procesos al resto de máquinas no puede acceder porque se lo impide el firewall del sistema.



3º *Máquina procesos.* Como se ve en el pantallazo siguiente cumple lo pedido, esta máquina solo ve a la máquina de bbdd, al resto de máquinas no puede acceder porque se lo impide el firewall del sistema.



4ª Máquina monitorización. Como se ve en el pantallazo siguiente cumple lo pedido, esta máquina accede a todas para poder monitorizarlas. El firewall le da acceso a todas las redes.



Como hemos visto en las imágenes, se cumplen las reglas del firewall y se ve como unas máquinas pueden ver a otras y viceversa y con esto concluimos viendo un ejemplo para poder instalar las máquinas de una empresa a través de openstack.

5. CONCLUSIONES.

Después de realizar todo el laboratorio hemos visto la flexibilidad que nos da Openstack en nuestra plataforma.

Una vez creada la plataforma podremos crear de una forma sencilla y cómoda nuestra estructura de máquinas para nuestro entorno. No tendremos que ser unos técnicos de sistema para crear las máquinas. Es verdad que la instalación de Openstack tiene una cierta complejidad y necesita administradores de sistema para su realización, pero una vez montada y gestionada por estos la gestión de máquinas virtuales y redes es mucho más sencilla para el usuario final. Esto que implica, pues que si eres una empresa grande con administradores de sistema, podrás dejarles a usuarios que monten sus máquinas para realizar sus proyectos, por ejemplo se le podrá dejar al equipo de desarrollo libertad para crear un entorno de pruebas ya sea para montar un nuevo proyecto o intentar reproducir un fallo de producción. En el caso de ser una empresa pequeña podrás subcontratar en la nube un entorno a tu medida sin necesidad de gasto de infraestructura ni de técnicos con conocimientos avanzados.

Openstack te da más beneficios, te da una flexibilidad de máquinas muy importante, imagínate que tienes una empresa en la nube con openstack, para un caso puntual necesitas el doble de máquinas que necesitas todos los días para un fin de semana, si has creado una plantilla puedes duplicar la potencia de tu empresa para lo necesario sin tener que sobredimensionar tu empresa. Esto en que te ayuda, principalmente en la reducción de costes, que es algo muy importante para cualquier empresa.

Que más beneficios da, pues otro principal, al estar soportado por tantas empresas y ser el producto de la nube más estándar te va a dar mucha facilidad de portabilidad, en qué sentido. Imaginemos que hemos contratado el servicio de cloud con una empresa y esta no da el servicio que nosotros queremos o es muy caro, pues si esta en openstack tendrás más posibilidades de portabilidad a otro entorno de forma fácil y sencilla que si esta en otra plataforma de cloud, ya que openstack al ser gratuita y tener tantas empresas detrás de ella es lo más parecido al estándar que hay en el mercado e intenta siempre estarlo.

Por estas razones este tipo de sofware tipo servicio están siendo tan utilizados y pedidos en el mercado laboral, por esta razón a los alumnos que ira dirigido este curso les servirá de gran ayuda cuando terminen la carrera en su futuro laboral, que es lo principal para una universidad.

6.REFERENCIAS.

Sobre software utilizado para la creación del laboratorio:

https://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube

http://www.monografias.com/trabajos-pdf5/cloud-computing/cloud-computing.shtml

https://wiki.debian.org/es/KVM

https://es.wikipedia.org/wiki/ISCSI

https://wiki.archlinux.org/index.php/Network_Time_Protocol_daemon_(Espa%C3%B1ol)

https://es.wikipedia.org/wiki/MariaDB

https://es.wikipedia.org/wiki/RabbitMQ

https://es.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol

Sobre openstack:

http://aprendiendoopenstack.blogspot.com.es/

https://openwebinars.net/que-es-eso-de-openstack-por-que-deberia-conocerlo/

https://www.sdxcentral.com/resources/open-source/what-is-openstack-quantum-neutron/

http://vmartinezdelacruz.com/en-pocas-palabras-como-funciona-openstack/

http://es.wikipedia.org/wiki/OpenStack#Block_Storage_.28Cinder.29

http://aprendiendoopenstack.blogspot.com.es/

https://wiki.openstack.org/wiki/Neutron/ML2

http://www.dbigcloud.com/cloud-computing/176-openstack-desde-cero-neutron-parte-1.html

http://www.dbigcloud.com/cloud-computing/176-openstack-desde-cero-neutron-parte-2.html

http://www-01.ibm.com/support/knowledgecenter/SS4KMC_2.2.0/com.ibm.sco.doc_2.2/sr/cg/cgt_cgcreavaz.html?lang=es

https://www.openstack.org/

7. PRESUPUESTO.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:

Jesús Sánchez Manzanero

2.- Departamento:

Ingeniería Telemática

3.- Descripción del Proyecto:

- Titulo	Diseño e implementación de un laboratorio de Openstack	
- Duración (meses)	6	
Tasa de costes Indirectos:		20%

4.- Presupuesto total del Proyecto (valores en Euros):

5275	Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a)]	Coste hombre mes	Coste (Euro)	Firma de conformidad
					0,00	
Jaime Jose Garcia Reinoso		Ingeniero Senior	1	4.289,54	4.289,54	
		Ingeniero		2.694,39	0,00	
					0,00	
					0,00	
		Hombres mes	**1**	**Total**	4.289,54	

[a)] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Portátil HP 15,6" Notebook 15-ac006ns Intel Core i5 5200U Ordenador Sobremesa HP Pavilion 500-515ns Intel Core i7 4790S	599,00	100	4	60	39,93
	999,00	100	4	60	66,60
		100		60	0,00
		100		60	0,00
		100		60	0,00
					0,00
				Total	106,53

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e)]

Descripción	Empresa	Costes imputable
	Total	0,00

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	4.290
Amortización	107
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	879
Total	5.275